VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

LABORATORNÍ ÚLOHY PRO VÝUKU SÍŤOVÝCH TECHNOLOGIÍ
LABORATORY EXERCISES FOR NETWORK TECHNOLOGIES EDUCATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Martin Kapusta**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Jan Dvořák**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Martin Kapusta ***ID:*** 170606
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Laboratorní úlohy pro výuku síťových technologií

POKYNY PRO VYPRACOVÁNÍ:

Diplomová práce spočívá v návrhu dvou nových laboratorních úloh včetně kompletních návodů vhodných pro studenty zahrnující výchozí scénář, doplňující úkoly, kontrolní otázky a vzorové řešení. Každá návržená úloha bude obsahovat podrobný popis návrhu. Nastudujte problematiku komunikačních protokolů a porovnejte možnosti dostupných simulačních prostředí. Po konzultaci s vedoucím jedno simulační prostředí zvolte a v něm laboratorní úlohy vypracujte. Při návrhu úloh se zaměřte zejména na některé z těchto oblastí: rozbor aplikačních protokolů, srovnání transportních protokolů, demonstrace funkce kvality služeb, porovnání různých linkových technologií, porovnání protokolů síťové vrstvy či směrovacích protokolů. Časová náročnost každé úlohy musí být přibližně dvě hodiny.

DOPORUČENÁ LITERATURA:

[1] FOROUZAN, B. A. TCP/IP Protocol Suite. Fourth edition, Boston: McGraw-Hill Higher Education, 2010, 979 stran. ISBN 978-0-07-337604-2.

[2] JEŘÁBEK, J. Komunikační technologie. Skriptum FEKT Vysoké učení technické v Brně, 2016. s. 1-172.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 16.5.2019

Vedoucí práce: Ing. Jan Dvořák
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Cieľom diplomovej práce je vybrať simulačné prostredie pre tvorbu laboratórnych úloh zameraných na výuku sieťových technológií. V teoretickej časti práce sú rozobraté základy sieťovej komunikácie, adresovanie, vrstvové modely a typy prenosu informácií. Popísané sú tiež technológie, na ktoré sú zamerané laboratórne úlohy. Sú to technológie Wi-Fi, Ethernet a smerovací protokol OSPF. V praktickej časti sa nachádza popis simulačných prostredí, z ktorých je jedno zvolené na tvorbu laboratórnych úloh. Konkrétne je to NS-3 simulátor. Pre obidve úlohy sú vypracované laboratórne návody, obsahujúce teoretický úvod, podrobný popis zdrojového kódu, samostatné úlohy, očakávané výstupy a kontrolné otázky zamerané na látku preberanú v laboratórnych úlohách.

KĽÚČOVÉ SLOVÁ

laboratórne úlohy, sieťové technológie, IP adresy, ISO/OSI, TCP/IP, sieťové simulátory, NS-3, Wi-Fi, Ethernet, OSPF

ABSTRACT

The aim of the diploma thesis is to choose network simulator suitable for network technologies laboratory tasks for educational use. Theoretical part of thesis describes basics of network communication, addressing, reference models. Thesis also describes standards Wi-Fi, Ethernet and routing protocol OSPF - technologies which are discussed in laboratory tasks. The practical part of diploma thesis describes a few available network simulators suitable for creating two laboratory tasks. Finally, the NS-3 simulator was chosen. Both laboratory tasks include theoretical introduction, detailed description of source code, individual tasks, expected outputs and control questions which senses understanding of discussed technologies.

KEYWORDS

laboratory tasks, network technologies, IP addresses, ISO/OSI, TCP/IP, network simulators, NS-3, Wi-Fi, Ethernet, OSPF

BIBLIOGRAFICKÁ CITÁCIA:

KAPUSTA, Martin. *Laboratorní úlohy pro výuku síťových technologií.* Brno, 2019. 139 s. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/118171. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Ing. Jan Dvořák.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Laboratórní úlohy pro výuku síťových technológií“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne …….. .....

podpis autora

POĎAKOVANIE

Rád by som poďakoval vedúcemu diplomovej práce pánovi Ing. Janu Dvořákovi za odborné vedenie, trpezlivosť, konzultácie a cenné rady pri spracovaní diplomovej práce.

Brno ....

(podpis autora)

OBSAH

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ÚVOD

Informačné technológie zmenili štýl života ľudstva. Používame mnohé technológie, prostredníctvom ktorých, sme schopní v krátkych časových úsekoch zdieľať dáta, informácie, prenášať zvuk, video. Veľký dôraz sa kladie na funkčnosť, kvalitu služieb, nepretržité fungovanie a reakciu informačných systémov na neočakávané udalosti.

Dôležitú úlohu zohrávajú rôzne sieťové technológie. Sieť je v podstate skupina hostí, ktorí sú navzájom poprepájaní medzi sebou a sú schopní vymieňať si informácie prostredníctvom káblových alebo rádiových spojení – pomocou určitých prenosových médií. Sieťou rozumieme malú sieť v domácnosti alebo firme až po siete, ktoré spájajú organizácie, mestá, štáty, kontinenty. Počítačovou sieťou nazývame aj dva počítače spojené navzájom medzi sebou.

Najväčšia sieť sa nazýva Internet. Je to komplexná sieť vlastnená rôznymi subjektmi, do ktorej sú popripájané koncové stanice. V tejto sieti sú použité rôzne sieťové technológie, prenosové média, zabezpečovacie technológie, aby bola zabezpečená správna a nepretržitá prevádzka s požadovanou kvalitou služieb. V súčasnosti sa väčšina služieb integruje do paketových alebo IP sietí. Pre tieto aj ďalšie dôvody je nutné, aby boli v tomto obore vzdelávaní odborníci, ktorí aj naďalej budú zabezpečovať chod sietí a podieľať sa na vytváraní nových technológií.

Výstupom diplomovej práce sú dve laboratórne úlohy, ktoré budú slúžiť na výuku sieťových technológií. V prvej časti sú popísané základné pojmy týkajúce sa komunikácie, techniky prenosu informácií a jednotlivých prvkoch siete. Ďalej sa v práci popisuje rozdelenie sietí podľa veľkostí. Rozobratý je model ISO/OSI, jednotlivé vrstvy a ich základné funkcie. Model ISO/OSI je porovnávaný s TCP/IP, popisované sú rozdiely a prípadné nedostatky alebo výhody jednotlivých modelov.

Práca obsahuje kapitoly, ktoré sa venujú popisu adresovania. Rozobraný je rozdiel medzi IPv4 a IPv6 protokolmi, IPv4 a IPv6 adresami, triednom rozdelení a úspore adresného priestoru pomocou techniky podsieťovania.

Nasledujúce kapitoly popisujú technológie Wi-Fi (štandard 802.11), Ethernet, techniky CSMA/CA, CSMA/CD a smerovací protokol OSPF, ktoré sú základnými technológiami použitými v laboratórnych úlohách.

V praktickej časti sú popísané dostupné simulačné prostredia na simuláciu sieťových technológií ich možnosti, výhody a nevýhody. Následne je zvolené jedno prostredie, konkrétne NS-3 a je navrhnutá štruktúra laboratórnych úloh. Rozpísaná je základná štruktúra kódu pri tvorbe skriptov v jazyku C++.

Ďalšia kapitola obsahuje popis skriptu pre obidve laboratórne úlohy s názvami „Technológia Wi-Fi a Ethernet“ a „Smerovací protokol OSPF“. Popisovaný je finálny skript po splnení všetkých samostatných úloh. Zobrazené sú očakávané výstupy po splnení laboratórnych úloh v podobe grafov, štatistík, trasovacích súborov, animácií a smerovacích tabuliek.

V prílohe sú uvedené laboratórne návody k obom úlohám, kde je popísaná tvorba základnej topológie, zadané sú samostatné úlohy a kontrolné otázky.

1 ÚVOD DO SIEŤOVÝCH TECHNOLÓGIÍ

1.1 Komunikácia

Základným účelom komunikácie je výmena informácií medzi dvoma účastníkmi. Jeden účastník vystupuje ako zdroj informácie, druhý ako prijímač informácie. Pri popise komunikácie v informatike je dôležité objasniť tri základné pojmy. Sú to informácie, dáta a údaje.

Dátami rozumieme akékoľvek informácie, ktoré sú spracovávané výpočtovými systémami alebo programami.

Údaje v informačných systémoch reprezentujú ich najnižší prvok, charakterizujú ho. Údajmi rozumieme najmenšie nedeliteľné prvky, hodnoty bez ohľadu na to či majú alebo nemajú informačný obsah. Informačný obsah je nulový ak daná informácia neprinesie nič nové. Patria sem písmená, znaky, čísla.

Informácie môžeme chápať ako správu, sú vhodné pre spracovanie človekom na prenos alebo zaznamenanie do pamäte.

Informácia, ktorá je prenášaná sa nazýva správa. Táto správa je prenášaná prenosovým kanálom. Príkladom prenosového kanálu sú telefónne linky, atmosféra, optické káble. Základnými parametrami sú šírka pásma udávaná v hertzoch Hz alebo bitoch za sekundu bit/s. Ďalším prvkom [1] je vysielač, ktorý danú správu vyšle do prenosového kanálu a následne prijímač, ktorý správu dokáže správne dekódovať. Obr. 1.1. Vysielač pridáva určitú redundanciu do správy kvôli zabezpečeniu proti chybám vzniknutým v dôsledku rušenia v prenosovom kanále. Redundancia tiež môže obsahovať kryptografické zabezpečenie. Na základe pridaných redundantných bitov protichybových kódov prijímač zisťuje, či bola správa prenesená bez chýb alebo nie.

Obr. 1.1 Bloková schéma dátovej komunikácie

1.2 Techniky prenosu informácií

Rozlišujeme štyri základné techniky prenosu informácií. Najvhodnejší spôsob prenosu dát sa volí na základe povahy prenášaného signálu. Dôležitým [2] faktorom pri rozhodovaní je nutnosť vedieť, či sa budú prenášať skôr dáta, kde je najdôležitejším kritériom spoľahlivý prenos a nezáleží vo veľkej miere na oneskorení alebo sa budú prenášať hovory, streamovať video a rôzne iné aplikácie v reálnom čase, kde je práve oneskorenie a jitter kritický. Medzi základné druhy prenosu dát patrí:

Komutácia okruhov (circuit switching) – prenosový kanál musí byť vytvorený pred samotným prenosom. Prenosový kanál je rezervovaný pre účastníkov po celú dobu ich aktivity. Pre polo-duplexný prenos je rezervovaný iba jeden kanál. Ak chceme prenášať dáta obojsmerne musia byť rezervované dva kanály – pre každý smer jeden. Tento [2] druh spojenia je drahý z hľadiska nákladov. Spojenie je udržiavané po celú dobu aj keď na linke nemusí dochádzať k prenosu informácií. Tento spôsob sa využíva hlavne pre prenos hlasových signálov.

Komutácia paketov (packet switching) – na rozdiel od komutácie okruhov v tomto druhu spojenie nie je potrebné vytvárať prenosový kanál a rezervovať jeho kapacitu na celú dobu spojenia. Prenosový [2] kanál je dostupný pre viacerých užívateľov. Ak je správa dlhá, musí sa rozdeliť na bloky dát, ktoré nazývame pakety. V paketových sieťach vznikajú problémy so správnym doručením paktu do cieľa. Ak je prekročená kapacita kanálu dôjde k jeho zahlteniu. Z týchto dôvodov sa používajú rôzne mechanizmy na zabezpečenie kvality služieb, správnemu prenosu a poskladaniu správy z jednotlivých paketov.

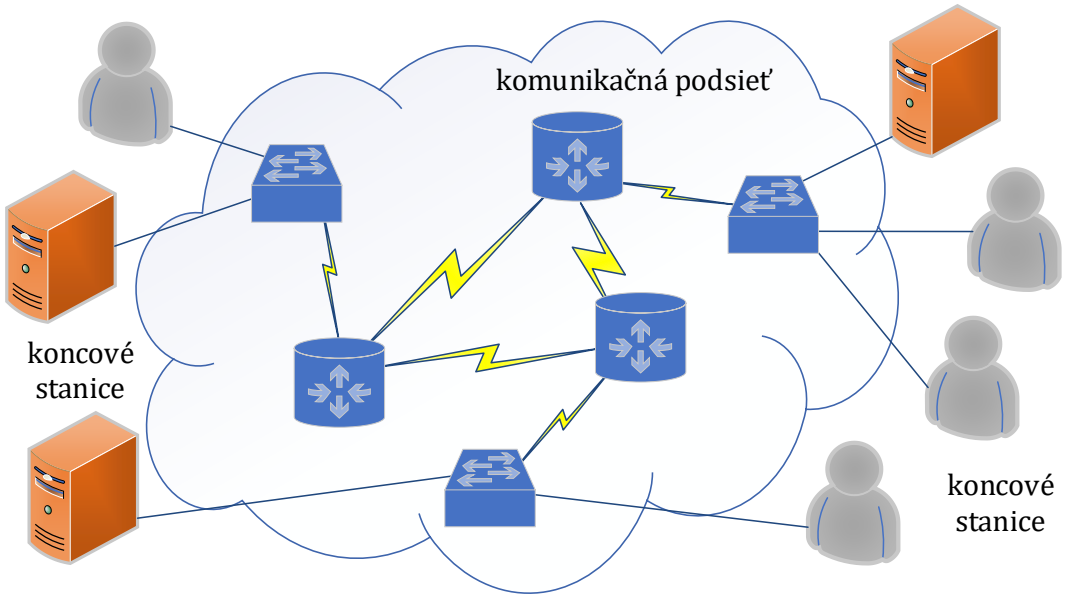
Komutácia správ (message switching) – pri tomto type spojenia sa nevytvára fyzické spojenie. Dáta sú prenášané zo zdroja dát k cieľovej stanici prostredníctvom rôznych uzlov siete. Princípom je, že správa sa pošle najbližšiemu uzlu, kde sa uloží a skontroluje. Následne po skontrolovaní môže byť poslaná do ďalšieho uzla na ceste k príjemcovi. Správy [3] sú malé samostatné jednotky

prenášané od odosielateľa po príjemcu, ktorí nemusia byť priamo spojení. Nevýhodou tohto typu sietí je kladenie veľkých nárokov na jednotlivé uzly, ktoré si musia jednotlivé správy ukladať – potrebujú veľkú kapacitu pamätí. Taktiež nemôžu byť používané pre aplikácie v reálnom čase alebo interaktívne aplikácie.

Komutácia buniek (cell switching) – je spojovaná s technológiou ATM (asynchronous transfer mode). Správa sa rozdelí na menšie celky o pevne stanovenej dĺžke. Následne sa pri prenose kontroluje len záhlavie. Tieto siete sú rýchle a dochádza k malému oneskoreniu prenosu. Používajú sa k prenosu rečových signálov aj klasických dát. Pri prenose [4] buniek sa zvyčajne používajú virtuálne okruhy, odozva týchto sietí je rýchla. Nevýhodou komutácie buniek je fixná veľkosť bunky. Technológia ATM používa bunky o dĺžke 53 bytov, kde 48 bytov tvorí správa a zvyšných 5 bytov obsahuje záhlavie.

1.3 Základné prvky sietí

Každá komunikačná sieť musí mať dva základné komponenty. Prvým sú prepojovacie prvky ktoré sa starajú o prenos, smerovanie, prepínanie. Druhým sú spoje, ktoré spájajú koncové stanice k prepojovacím prvkom. Základná schéma je na obrázku Obr. 1.2.



Obr. 1.2 Základná štruktúra siete

Komunikačná sieť je skupina dvoch a viac koncových staníc, ktoré sú medzi sebou spojené a zdieľajú spoločné prostriedky. Medzi základné komponenty komunikačných sietí patrí kabeláž, prepínače, smerovače, brány, mosty.

Prenosové média slúžia na prenášanie signálov medzi zariadeniami. Prenosové média sa delia na metalické, optické a rádiové. Z metalických vedení sa používajú krútené páry. Je to symetrické vedenie, ktoré je rozdeľované do kategórií 1, 2, 3, 4, 5, 5E, 6 a 7. Kategórie 5E, 6, 7 sú vysokorýchlostné krútené páry, ktoré prenášajú rýchlosti 1 Gbps a vyššie. Vyvinuté boli z telefónnych rozvodov, používajú sa v LAN (Local Area Network), dosah je 100 m. Kábel ma konektor RJ-45, [5] označovaný je UTP (Unshielded Twister Pair) alebo vo variante zatienenej voči rušeniu zvonku STP (Shielded Twister Pair).

Z nesymetrických metalických vedení sem zaraďujeme koaxiálne káble. Koaxiálne káble [6] sú drahšou variantou krútených párov, základné prenosové pásmo pri koaxiálnych kábloch bez použitia modulácie je od 0-150 MHz, pri použití modulácie je to 50-750 MHz. Dosah tohto typu vedenia sú stovky metrov.

Optické vlákna majú vysokú prenosovú kapacitu, sú odolné voči rušeniu a odpočúvaniu. Využívajú sa v nich zákony optiky, zákon odrazu, zákon lomu, index lomu. Na dlhšie prenosové trasy sa používa vlnový multiplex. Delíme ich na jednovidové a viacvidové. Jednovidové optické vlákna majú väčšie prenosové rýchlosti a vyššiu cenu oproti viacvidovým. Preto je výhodne použiť viacero vlákien v jednom optickom kábli. Rádiové spoje sú založené na prenose elektromagnetických vĺn voľným priestorom. Majú veľký dosah a sú schopné prejsť budovami. Sú závislé na frekvenciách. Rádiové [6] vlny majú veľké dosahy, preto je dôležité koordinovať ich frekvencie. Vlny na nízkych kmitočtoch dobre prechádzajú cez prekážky. Vlny na vyšších kmitočtoch sa od prekážok odrážajú a sú tým pádom závisle aj od poveternostných podmienok, používajú sa v interiéri.

Prepínač je zariadenie podobné rozbočovaču, ktoré má navyše viacero funkcií. Obsahuje väčší počet portov a vie doručovať správy konkrétnym portom na základe fyzických adries nazývaných MAC adresy (Media Access Control Address). Prepínač nezaťažuje zbytočne sieť posielaním správ do celej siete ale obmedzuje sa na konkrétny port, čo zvyšuje rýchlosť siete. Obsahuje [7] niekoľko

portov, napájanie a LED diódy znázorňujúce aktivitu jednotlivých portov. Nachádza sa na druhej vrstve ISO/OSI modelu.

Smerovač sa nachádza na tretej vrstve ISO/OSI modelu. Má za úlohu smerovanie v sieti a medzi sieťami. Ak smerovač príjme paket, v záhlaví zistí IP adresu a na základe smerovacích tabuliek vytvorených staticky alebo dynamicky zvolí cestu, kadiaľ paket odošle. Smerovače sú protokolovo závislé. Smerovače sa delia na viacero kategórií. Sú to malé smerovače [8] v domácnostiach. Príkladom je DSL smerovač, ktorý pripája užívateľa k internetu cez ISP (Internet service provider)–poskytovateľa internetu. Najvyššie v hierarchii smerovačov sú smerovače tvoriace chrbtovú sieť (Internet Backbone). Je to vysokorýchlostná sieť posielajúca dáta prostredníctvom optických spojov medzi mestami, štátmi a kontinentmi.

Firewall býva realizovaný buď softvérovo alebo kombináciou softvérového a hardvérového riešenia. Firewall sa používa na ochranu siete z vonku proti útočníkom aj z vnútra na riadenie prístupu. Funguje na princípe hierarchicky usporiadaných presne definovaných pravidiel. Najpoužívanejšími firewallmi sú paketový a stavový firewall. Paketový firewall je založený na filtrovaní paketov. Pakety sú identifikované IP adresou a typom služby v UDP alebo TCP záhlaví. Stavový firewall [9] zisťuje IP adresy, porty a zachováva si tabuľku povolených spojení. V tabuľke sú zaznamenané údaje, ktoré charakterizujú dané spojenie – IP adresy, príznaky alebo potvrdzovacie čísla segmentov.

Východzia brána má za úlohu spájať rozdielne siete, ktoré pracujú s rôznymi protokolmi. Má funkciu východzej brány aj smerovača. Pakety, [10] ktoré nenájdu adresáta v lokálnej sieti, smerujú do inej siete alebo podsiete, sú na východziu bránu smerované. Môžeme ju nazvať vstupom do siete. Brány môžu fungovať na sieťovej, transportnej alebo aplikačnej vrstve.

1.4 Delenie sietí podľa veľkosti

Siete sa najčastejšie delia podľa veľkosti. V každom type sietí sa používajú rôzne technológie tak, aby splnili požiadavky kladené na konkrétny typ siete. Podľa oblasti, ktorú pokrývajú ich delíme na: PAN (Personal Area Network), LAN (Local

Area Network), MAN (Metropolitan Area Network), WAN (Wide Area Network). Ďalej poznáme EPN (Enterprise Private Network), VPN (Virtual Private Network).

PAN (Personal Area Network) je sieť využívaná malým počtom užívateľov. Má nižšie prenosové rýchlosti. Rádovo sú to jednotky Mbit/s. Patria sem [11] siete s tlačiarňami, skenermi, tabletmi, smart telefónmi, bezdrôtové technológie Bluetooth, IrDa (Infrared Data Associattion) a podobné technológie. Ako príklad môžeme uviesť prenos medzi dvoma telefónmi, alebo počítačom a tlačiarňou.

LAN (Local Area Network) spájajú počítače a pracovné stanice. Lokálne siete sa väčšinou nachádzajú v jednej budove alebo viacerých budovách maximálne do vzdialenosti niekoľkých kilometrov. Z prenosových médií prevládajú krútené páry a koaxiálne káble. Z dôvodu [12] krátkych vzdialeností môžeme zanedbať rušenie. Rýchlosti sa pohybujú od 100 Mbit/s, 1 Gbit/s po 10 Gbit/s. Oneskorenie [13] v LAN sieťach je typicky od 10 μs po 1 ms. Najpoužívanejšie technológie sú Fast Ethernet a Gigabit Ethernet.

MAN (Metropolitan Area Network) je sieť väčšia ako LAN. WAN siete väčšinou spájajú viacero LAN sietí, ktoré môžu byť v iných mestách. Na ich princípe sú vybudované aj národné siete. Sú vybudované [13] s dôrazom na vysokú rýchlosť. Slúžia na prenos dát, audio správ, video správ. Rýchlosti v MAN sieťach sa pohybujú na úrovni 1 Gbit/s a viac. Medzi bežne používané technológie patria optické vlákna, prostredníctvom ktorých je realizovaný Fast Ethernet. Oneskorenie je nízke od 100 μs po 10 ms. Siete typu [13] MAN sú väčšinou spravované určitou organizáciou avšak poskytovaná je viacerým subjektom.

WAN (Wide Area Network) je sieť, ktorá pokrýva rozľahlé oblasti. Môže pokrývať až tisícky kilometrov vzdialené miesta. Technológie používané vo WAN sieťach sú vysokorýchlostné, avšak vo všeobecnosti pomalšie ako v sieťach LAN, MAN. Používajú sa napríklad MPLS (Multiprotocol Label Svitching), ATM (Asynchronous Transfer Mode), FR (Frame Relay) a ďalšie. [15] Prenosové kapacity sú tiež zdieľané, a sú vlastnené rôznymi organizáciami. Prenosová kapacita sa prenajíma. Vo WAN sieťach sa nachádza veľký počet serverov, terminálov a dáta sa prenášajú na obrovské vzdialenosti. Rýchlosť sa pohybuje od kbit/s po Mbit/s. Na WAN site sú kladené vysoké požiadavky. Kritické je [14] oneskorenie, ktoré musí byť čo najmenšie. Pohybuje sa rádovo od jednotiek ms po stovky ms.

2 VRSTVOVÝ MODEL A ADRESOVANIE

Sieťová komunikácia je založená na vrstvách. Aby sa dali jednotlivé siete prepojiť, bolo potrebné definovať určité referenčné modely komunikácie. Takto sa zamedzilo nekompatibilite sietí s uzavretou architektúrou a bolo možné jednotlivé siete začať prepájať.

Medzi dva základné modely vrstvovej komunikácie patrí ISO/OSI (Open System Interconnection Reference Model). Skladá sa zo siedmich vrstiev, kde každá vrstva zastáva určité úlohy.

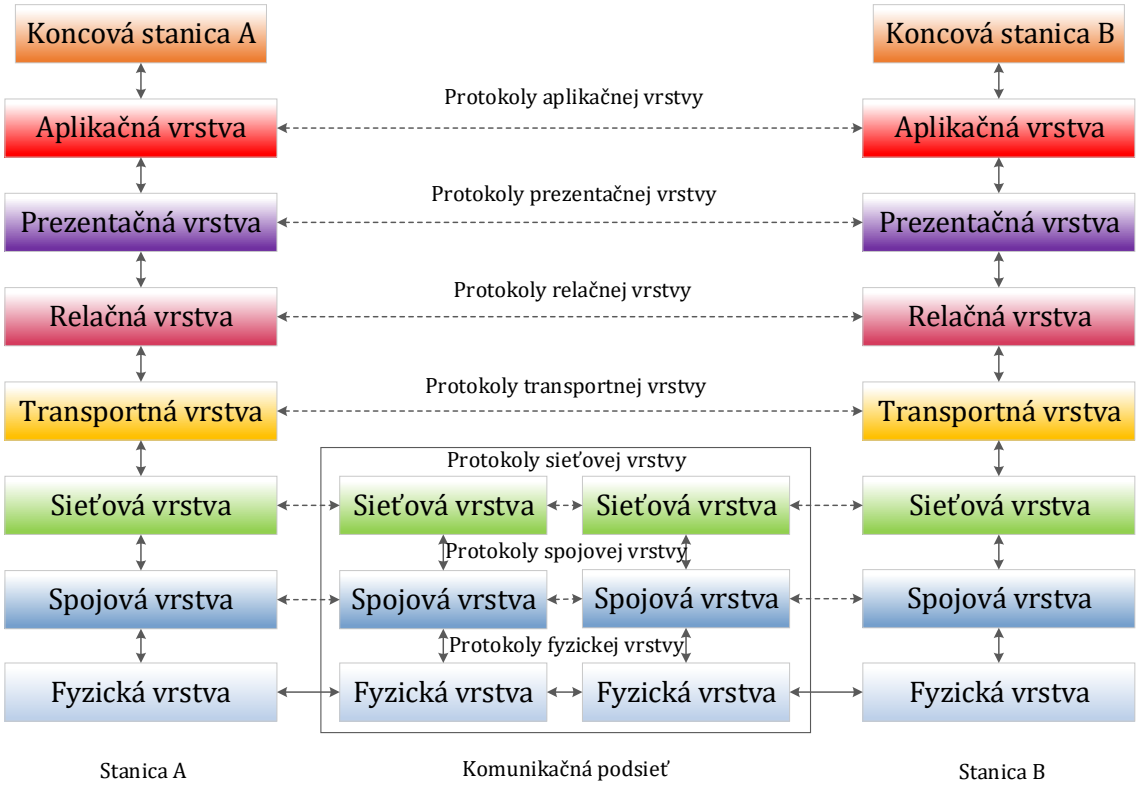
Druhým modelom je model TCP/IP (Transmission Control Protocol/Internet Protocol). TCP/IP zahrnuje množstvo protokolov a je rozčlenený na štyri vrstvy. Architektúra TCP/IP zlučuje funkcie niektorých vrstiev modelu ISO/OSI. V mnohých prípadoch je vhodnejšie implementovať architektúru TCP/IP.

2.1 Vrstvový model ISO/OSI

Vrstvový model ISO/OSI je referenčný model, ktorý slúži na realizáciu komunikácie medzi dvomi sieťovými systémami. Model ISO/OSI rozdeľuje komunikáciu do siedmich vrstiev. Každá z vrstiev [15] má určité úlohy a používa určitú sadu protokolov.

Každá vrstva sa spolieha na služby vrstvy, ktorá je hierarchicky nižšie a poskytuje služby vrstve, ktorá je hierarchicky vyššie. Jednotlivé vrstvy sú: fyzická, spojová, sieťová, transportná, relačná, prezenčná a aplikačná. V štandarde nie je presne definovaná implementácia, ale sú tu definované základné princípy, účely jednotlivých vrstiev a služby, ktoré má daná vrstva poskytovať vyššej vrstve a vyžadovať od nižšej vrstvy.

Na každej vrstve sa postupne k dátam pridáva hlavička konkrétnej vrstvy – dáta sa zapuzdrujú. Pri prijatí správy príjemcom, každá vrstva hlavičku jej prislúchajúcu rozbalí a posiela dáta vyššej vrstve. ISO/OSI model pozostáva z nasledujúcich siedmich vrstiev zobrazených na obrázku Obr. 2.1.



Obr. 2.1 Architektúra siete podľa modelu ISO/OSI

Fyzická vrstva (Physical layer) - je najnižšia z vrstiev. Má za úlohu starať sa o fyzickú konektivitu. Vrstva definuje konektory, prenosové média, kapacity kanálov ich fyzikálne vlastnosti. Je zodpovedná za prenos a konvertovanie digitálnych signálov na analógové. Musí synchronizovať [16] bitový tok medzi prijímačom a vysielačom. Základnou požiadavkou na fyzickú vrstvu je čo najmenšie zanášanie chýb do bitového toku pri prenose a efektívny rýchly prenos signálov.

Spojová vrstva (Data Link Layer) - synchronizuje dáta prenášané fyzickou vrstvou. Jednou z hlavných funkcií spojovej vrstvy je zaručiť, že prenesené dáta prostredníctvom fyzickej vrstvy sú bezchybné a ak nie sú, musí ich opraviť. Na spojovej vrstve prebieha tiež ochrana proti zahlteniu a riešenie konfliktov pri viacnásobnom prístupe k médiu.

Delí sa [16] na dve podvrstvy LLC (Logical Link Control) a MAC (Media Access Control). Rámce sa na tejto vrstve radia, zisťuje sa, ktoré chyby nie je možné opraviť a riadi sa prístup k prenosovému médiu. Medzi základné protokoly patria

ARP (Address Resolution Protocol) – používa sa na hľadanie fyzickej adresy v lokálnej sieti na základe IP adresy, RARP (Reverse Address Resolution Protocol) – používa sa na hľadanie IP adresy na základe fyzickej adresy.

Sieťová vrstva (Network Layer) - základnou úlohou sieťovej vrstvy je smerovanie. Na sieťovej vrstve sa pracuje s paketmi. Nutné je zabezpečiť komunikáciu medzi jednotlivými smerovačmi. Sieťová vrstva musí riešiť problém nekonzistentnosti siete, jej rozličné vlastnosti a použité technológie. Zabezpečuje preklad fyzických adries druhej vrstvy na logické adresy (IP adresy). Kontroluje [17] bezchybnosť prenesených paketov, kontroluje tok dát aby nedošlo k zahlteniu. Sú v nej definované rôzne protokoly, ktoré sa starajú o smerovanie. Dôležitou úlohou je zabezpečovanie kvality služieb.

Transportná vrstva (Transport Layer) - základnými funkciami sú multiplexovanie, segmentácia a zabezpečovanie dostatočnej kvality spoju. Transportná vrstva je veľmi komplexná a služby, ktoré poskytuje vo vysokej miere závisia na požiadavkách siete. Správu delí na segmenty. Ako [18] každá vrstva aj transportná vrstva zabezpečuje riadenie chybových stavov. Najbežnejšie protokoly transportnej vrstvy sú protokoly TCP (Transmission Control Protocol)– zabezpečuje spoľahlivý prenos a UDP (User Datagram Protocol)–nezabezpečuje spoľahlivý prenos. Všetky používané protokoly sú koncového charakteru.

Relačná vrstva (Session Layer) - synchronizuje komunikáciu medzi dvoma rozličnými systémami a spolupracujúcimi aplikačnými procesmi. Prezentačná [16] vrstva poskytuje relačnej vrstve vytváranie a ukončovanie relácie, oznamovanie výnimočných situácií, pozdržanie prenosu správ. Zamedzuje, aby sa kritické operácie vykonávali naraz. Vrstva je schopná pridávať synchronizačné známky po určitom počte stránok pre jednoduchšie overovanie správnosti poslaných dát.

Prezentačná vrstva (Presentation Layer) - stará sa, aby bol príjemca schopný dáta dekódovať, stará sa o kompresiu dát, šifrovanie, transformáciu kódovania, pripravuje dáta pre aplikačnú vrstvu. Prekladá dáta do formátu, ktorý

vyžaduje koncová stanica. Môže priamo zasahovať to užívateľských dát a od relačnej vrstvy požaduje prenos týchto dát.

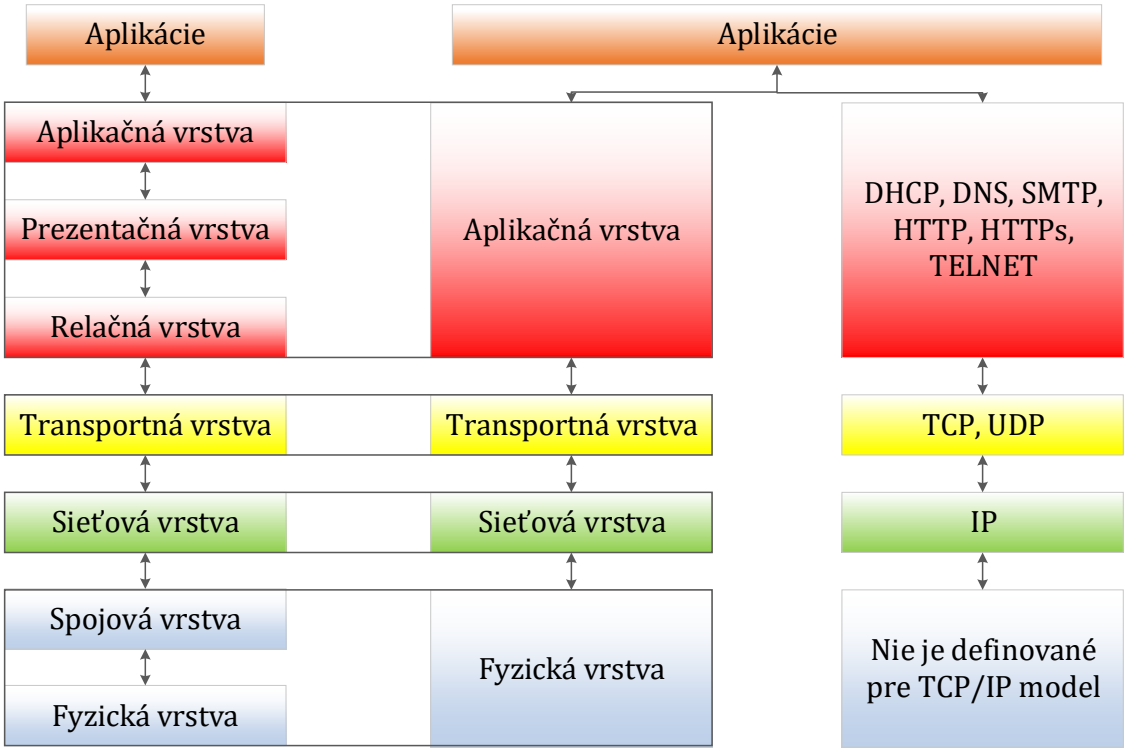
Aplikačná vrstva (Application Layer) - je najvyššou vrstvou. Umožňuje [19] [20] užívateľským aplikáciám prístup k sieti. Obsahuje množstvo protokolov, ktoré poskytujú mailové služby, služby prenosu súborov, prehliadania stránok, vyhľadávania doménových mien, prístupu k vzdialeným zariadeniam, konfiguráciu IP adries, prenos súborov, signalizáciu, zabezpečený prenos dát, ukladanie pošty, prístup do vzdialených databázových systémov a rôzne iné služby potrebné pre správny chod užívateľských aplikácií. Tiež zaisťuje informácie o stave komunikátorov, dohaduje ochranu správ, určuje kvalitu služieb. Zabezpečuje synchronizáciu a pomáha pri určovaní syntaxe. Je náročná na prostriedky koncových staníc.

2.2 Vrstvový model TCP/IP

Vychádza z modelu ISO/OSI. V súčasnosti sa používa vo všetkých operačných systémoch a je využívaný pre komunikáciu cez sieť Internet. Obsahuje štyri vrstvy. Myšlienkou TCP/IP modelu je dosiahnuť nezávislosť na prenosových médiách. Model TCP/IP predpokladá iba nespoľahlivé služby na nižších vrstvách. O spoľahlivosť sa starajú vyššie vrstvy ak sú požiadané o túto službu.

V modeli TCP/IP sú viaceré funkcie prenesené na koncové stanice. Týmto krokom sa značne zníži zaťaženie komunikačnej siete. Komunikačná sieť však nesmie zahadzovať pakety bezdôvodne. Vyvíja maximálnu snahu (služba best effort) na doručenie paketov. Zahadzuje ich len v hraničných prípadoch ako je poškodenie paketu, zahltenie siete, výpadok spojenia. Tieto opatrenia kladú vysoké nároky na koncové stanice.

Štruktúra TCP/IP modelu je zobrazená na obrázku Obr. 2.2. Pozostáva z fyzickej, sieťovej, transportnej a aplikačnej vrstvy.



Obr. 2.2 Porovnanie modelu ISO/OSI a TCP/IP

Vrstva sieťového rozhrania (Network Interface Layer) - zaisťuje prenos rámcov medzi priamo pripojenými koncovými stanicami. Táto vrstva [21] spája funkcie fyzickej a spojovej vrstvy definovanej v IOS/OSI modeli. TCP/IP model vrstvu sieťového rozhrania bližšie nešpecifikuje, pretože je závislá od použitej technológie.

Internetová vrstva (Internet Layer) - má funkcie ako sieťová vrstva z ISO/OSI modelu. Používajú sa tu protokoly IP (Internet Protocol) a ICMP (Internet Control Message Protocol). IP protokol poskytuje iba nespoľahlivú službu, ktorá má nespojový charakter. IP protokol [22] musí preklenúť rozdiely jednotlivých sietí.

Transportná vrstva (Transport Layer) - je ekvivalentom transportnej vrstvy z ISO/OSI modelu.

Aplikačná vrstva (Application Layer) - spája funkcie relačnej, prezenčnej a aplikačnej vrstvy ISO/OSI modelu. Sú [21] na nej spustené všetky aplikácie v rámci modelu TCP/IP. Tieto aplikácie komunikujú priamo s transportnou vrstvou. Všetky relačné a prezentačné služby si zaisťujú jednotlivé aplikácie. Výhodou je, že ak aplikácia relačné a prezenčné služby nepotrebuje nevzniká zbytočná réžia a sieť je menej zaťažená. Na tejto vrstve sa nachádza veľké množstvo protokolov.

2.3 Adresácia

Komunikácia v sieti sa môže uskutočniť len za predpokladu, že sa jednotlivé prvky siete vedia medzi sebou identifikovať. Pri smerovaní medzi sieťami je nutné jednotlivým zariadeniam prideliť logické adresy, ktoré sa nazývajú IP adresy (Internet Protocol address). IP adresy sa rozdeľujú na IPv4 adresy a IPv6 adresy.

2.4 IPv4 adresy

IPv4 adresa ma dĺžku 4B čo je 32b. Môžeme ju zapísať v binárnom formáte alebo dekadickom. Každý oktet je oddelený bodkou. Napríklad IPv4 adresa 193.12.99.18 je ekvivalentná k zápisu 11000001.00001100.01100011.00010010. [23] Každá IP adresa musí jednoznačne identifikovať sieť a stanicu v danej sieti. IP adresy verzie 4 sú v dnešnej dobe vyčerpané. Na začiatku sa IPv4 adresy delili na triedy čo tento problém prehĺbilo. V súčasnosti [24] sa problém s vyčerpaním rozsahu IP adries zmiernil pomocou podsieťovania. Z triedneho rozdelenia poznáme triedy A, B, C, D, E. Jednotlivé rozsahy sú uvedené v tabuľke tab. 1.1.

Tab. 1.1 Triedne rozdelenie IPv4 adries

Trieda	Rozsah prvého bajtu	Maska	Počet sietí	Počet hostí
A	0 - 127	255.0.0.0	128	16777214
B	128 - 191	255.255.0.0	16383	65534
C	192 - 223	255.255.255.0	2097150	254
D	224 - 239	255.255.255.255	-	-
E	240 - 255	-	-	-

IP adresa sa skladá zo 4B, ktoré sú rozdelené na prefix a sufix. Na zistenie prefixu sa používa sieťová maska. Maska je binárny reťazec s dĺžkou 32 bitov. Prvá časť sa skladá s jednotiek druhá časť s núl. Adresu siete [24] v ktorej sa nachádza cieľová adresa získame logickým vynásobením adresy siete a masky. Príklad je uvedený v tabuľke tab. 1.2.

Tab. 1.2 Ukážka zistenia adresy siete pomocou masky

Adresa stanice dekadicky	193	12	99	18
Adresa stanice binárne	11000001	00001100	01100011	00010010
Maska siete dekadicky	255	255	255	0
Maska siete binárne	11111111	11111111	11111111	00000000
Adresa siete dekadicky	193	12	99	0
Adresa siete binárne	11000001	00001100	01100011	00000000

Z tabuľky tab. 1.2 vyplýva, že adresa siete je 193.12.99.0/24. IP adresy, ktoré môžeme použiť v danej sieti, sú v rozsahu od 193.12.99.1 do 193.12.99.254. Všesmerová adresa je posledná adresa z rozsahu v tomto prípade 193.12.99.255.

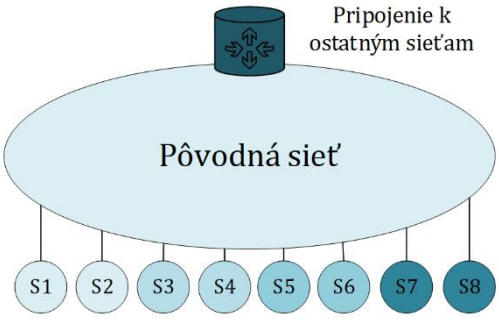
2.4.1 Podsiete

Podsiete umožňujú zefektívniť využívanie IPv4 adries rozdeľovaním veľkých sietí na menšie celky. Základnou [24] myšlienkou je použitie prvých k bitov pre definovanie podsiete a následne zostávajúcich (n-k) bitov na adresáciu stanice. Napríklad adresu 193.12.99.18 rozdelíme na štyri rovnako veľké podsiete. Podsiete sa líšia v poslednom bajte IP adresy. Maska bude mať hodnotu /26 namiesto pôvodnej hodnoty /24. Prefix bude mať dĺžku 26 bitov sufix 24 bitov. Začiatočné bity posledného bajtu IP adresy budú 00, 01, 10, 11. Zostávajúcich 6 bitov z daného posledného bajtu bude použitých na adresáciu koncových staníc. V každej podsieti tak vznikne priestor 2^6-2 = 62 staníc. Takýmto spôsobom sa získajú nezávislé adresné rozsahy, tým sa ušetrí adresný priestor.

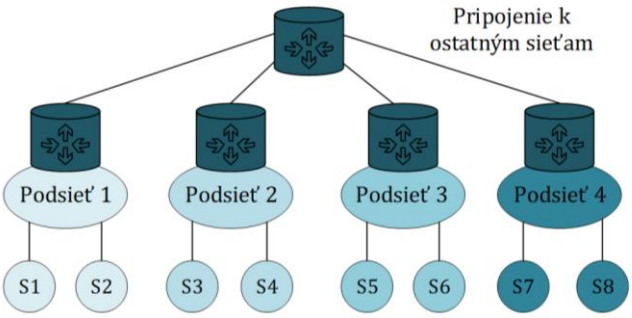
Tab. 1.3 Ukážka podsieťovania

Adresa	1.bajt	2.bajt	3.bajt	4.bajt	
Pôvodná sieť dekadicky	193	12	99	stanica	
binárne	11000001	00001100	01100011	00000000	
1.subsieť dekadicky	193	12	99	00	stanica
binárne	11000001	00001100	01100011	00	stanica
2.subsieť dekadicky	193	12	99	01	stanica
binárne	11000001	00001100	01100011	01	stanica
3.subsieť dekadicky	193	12	99	10	stanica
binárne	11000001	00001100	01100011	10	stanica
4.subsieť dekadicky	193	12	99	11	stanica
binárne	11000001	00001100	01100011	11	stanica

Z tabuľky tab. 1.3 vyplýva, že z jednej siete 193.12.99.0/24 sa vytvorili štyri menšie siete. Sú to siete 193.12.99.0/26, 193.12.99.64/26, 193.12.99.128/26, 193.12.99.192/26. V každej z nich je teda možné adresovať 62 koncových staníc. Výsledok podsieťovania je zobrazený na obrázku Obr. 2.3. Pôvodná sieť je zobrazená na obrázku Obr. 2.4. Z obrázkov je vidieť rozdelenie pôvodnej IP adresy na štyri menšie siete, čo znamená úsporu adresného priestoru.



Obr. 2.4 Pôvodná sieť



Obr. 2.3 Výsledok podsieťovania

2.5 IPv6 adresy

Protokol IPv6 vznikol v dôsledku vyčerpávania adresného rozsahu IPv4 protokolu na začiatku 90. rokov. S narastajúcim množstvom zariadení sa tento problém ešte viac prehĺbil. IPv4 protokol má 2^{32} adries na rozdiel od IPv6 protokolu, ktorý ich má 2^{128} čo predstavuje reálne nevyčerpateľný adresný priestor. Toto číslo si môžeme predstaviť ako hodnotu 1564 adries na meter štvorcový Zeme. IPv6 adresy [25]

sú dlhé, preto sa zvolil hexadecimálny zápis. Je vo formáte ôsmich hexadecimálnych číslic oddelených dvojbodkou. Príklad IPv6 adresy:

2001:0db8:0000:0000:0000:0000:1428:57ab

Skrátene sa môže zapísať nahradením bloku núl „::“aj ako

2001:0db8::1428:57ab je ekvivalentná **2001:db8::1428:57ab**

Počet zariadení pripájaných k internetu exponenciálne rastie a je dôležité postupne nasadzovať protokol IPv6. IPv4 a IPv6 protokoly nie sú navzájom kompatibilné a je nutné používať rôzne iné protokoly na preklenutie rozdielov pri prenose cez sieť. Protokol IPv6 má efektívnejšiu, jednoduchšiu hlavičku, nesie len najnutnejšie informácie a zameriava sa hlavne na služby poskytované sieťovou vrstvou. Hlavička IPv6 protokolu je dvojnásobne veľká oproti IPv4 avšak adresa je štyrikrát dlhšia. Tieto dva protokoly musia bežať súbežne, pretože sa nedá vypnúť IPv4 protokol v jednu chvíľu ale musí byť nahradený postupne.

Myšlienkou IPv6 adries je návrat ku koncovej komunikácii (end-to-end), kde by malo každé zariadenie IP adresu bez prekladov privátnych adries na verejné pomocou NAT. IPv6 má pripravené mechanizmy na zaisťovanie QoS, prináša nové protokoly ICMPv6 (Internet Control Management Protocol version 6) a DHCPv6 (Dynamic Host Control Protocol version 6), prepracovaný mechanizmus mobility staníc, ktorá je dôležitá pri mobilných zariadeniach, aby im zostala rovnaká IP adresa pri pohybe, podpora multicastového vysielania. Zjednodušenie prináša aj z pohľadu fragmentácie, ktorá už nebude pri IPv6 prebiehať. Stretneme [26] sa aj s zredukovaním smerovacích tabuliek.

Rozlišujeme tu adresy individuálne (unicast)-sú identifikátormi jednotlivých rozhraní aby sa k ním doručili pakety, skupinové (multicast)-používajú sa pre adresáciu skupín, pakety sa doručujú všetkým členom v skupine, majú byť náhradou všesmerovej adresy, a výberové (anycast)-sú podobné ako skupinové s tým rozdielom, že pakety sa posielajú jednému zo skupiny, väčšinou tomu čo je najbližšie k smerovaču od ktorého prišiel daný paket. Z hľadiska [26] bezpečnosti je výhodná relatívna anonymita koncových staníc. Nevýhodou je tiež to, že pri IPv4 adresách je vyriešených mnoho problémov s bezpečnosťou, ktorých riešenia pri protokole IPv6 nie sú doteraz jasné. Tiež bolo nutné upraviť softvér a hardvér pri nasadzovaní IPv6. Dôležité je zabezpečenie komunikácie a funkčnosti všetkých služieb po dobu

prechodu z IPv4 na IPv6. Operačné [27] systémy podporujú tento protokol istú dobu, pretože v súčasnosti nie je možné bez znalosti IPv6 fungovať.

2.6 Prechod z IPv4 na IPv6

Okamžitý prechod na IPv6 nie je možný, pretože IPv6 nie je kompatibilný spätne s IPv4. Nové protokoly nie je možné implementovať do niektorých starších technológií, preto musí byť prechod na IPv6 sieť plynulý a postupný. Niektoré [27] siete obsahujú iba IPv4 protokoly, alebo iba IPv6 a niektoré ich kombinujú. Existujú techniky, ktoré sa v súčasnosti používajú. Cieľom týchto techník je zabezpečiť plynulý prechod na sadu IPv6, aby mohol byť protokol IPv4 úplne zastavený.

Súbeh protokolov (Dual Stack) je technika, pri ktorej sú smerovačom podporované IPv4 a IPv6 protokol. Sú od seba nezávislé. Celá komunikácia siete pre IPv6 prebieha pomocou protokolu IPv6 a komunikácia protokolu IPv4 prebieha prostredníctvom IPv4 siete. Protokoly [28] sa navzájom neovplyvňujú – stanice musia mať obe adresy. Toto riešenie je nákladné avšak používa sa vo veľkej miere. Veľké množstvo sietí organizácii funguje týmto spôsobom.

Tunelovanie (Tunneling) predstavuje prenos paketov prostredníctvom zapúzdrenia do druhého protokolu. Ak máme [28] sieť IPv4 a chceme prostredníctvom nej preniesť pakety protokolu IPv6, tak na hraničnom smerovači od odosielateľa prebehne jeho zapúzdrenie do paketu IPv4 a na bráne siete príjemcu prebehne jeho rozbalenie. Následne pokračuje ako paket IPv6. Pri zapúzdrovaní nastáva zbytočná réžia a môžu nastať chyby.

Preklad adries využíva NAT64 (Network Address Translation 64), pri ktorom sa mení IPv4 adresa na IPv6 adresu alebo opačne záleží na požiadavkách siete. Ak chce stanica komunikovať prostredníctvom IPv6 siete protokolom IPv4 tak je na hraničnom smerovači odstránené záhlavie IPv4 a paketu je pridané záhlavie IPv6. Následne je tento paket prenášaný IPv6 sieťou. Pri jeho návrate k odosielateľovi táto služba prebehne reverzne. Znamená [28] to, že IPv6 záhlavie bude odstránené a paket dostane pôvodné IPv4 záhlavie a bude smerovaný k odosielateľovi. Takéto zmeny v paketoch vytvárajú problémy pri určitých spojeniach a nemusia spolupracovať s niektorými technológiami.

3 POPIS ĎALŠÍCH TECHNOLÓGIÍ SÚVISIACICH S RIEŠENÝMI LABORATÓRNYMI ÚLOHAMI

3.1 Bezdrôtové siete Wi-Fi

Wi-Fi technológia bola vyvinutá v Holandsku v roku 1991. Technológia je určená pre mobilné zariadenia. Používa sa na miestach, kde je potrebné sprístupniť sieťové služby užívateľom s mobilnými koncovými stanicami. Budovanie sietí, v ktorých je potrebná mobilita, je základnou motiváciou na používanie technológie Wi-Fi. Tento štandard umožňuje komunikáciu v LAN (Local Area Network) bez použitia káblov. Je súčasťou štandardov IEEE 802 a označuje sa ako IEEE 802.11. Sieťový adaptér prostredníctvom antény vysiela dáta vo forme rádiového signálu, ktorý je následne prijímaný Wi-Fi prijímačmi ako sú napríklad počítače, smartfóny a iné. Wi-Fi karta príjme signál a následne vytvorí spojenie medzi koncovou stanicou a užívateľom.

3.1.1 Komponenty Wi-Fi sietí

Architektúra siete podľa štandardu 802.11 obsahuje štyri základné komponenty. Patria sem [29] stanice, prístupové body, bezdrôtové prenosové médium a distribučný systém.

Stanice sú zariadenia, ktoré disponujú bezdrôtovým sieťovým rozhraním. Tieto zariadenia môžu byť buď mobilné alebo desktopové. Pri desktopových zariadeniach sa bezdrôtové spojenia využívajú na tvorbu bezdrôtových LAN sietí. Takýto postup sa používa vo veľkých otvorených oblastiach. Pri zavádzaní nového zariadenia nie je nutnosť pridávať ďalší vodič.

Prístupové body slúžia na konverziu rámcov na rámce, ktoré môžu byť ďalej rozosielané pomocou bezdrôtovej siete. Prístupové body sú bodom medzi bezdrôtovou sieťou a klasickou sieťou. Prístupový bod vysiela periodicky Beacon rámce, ktoré obsahujú informácie o sieti.

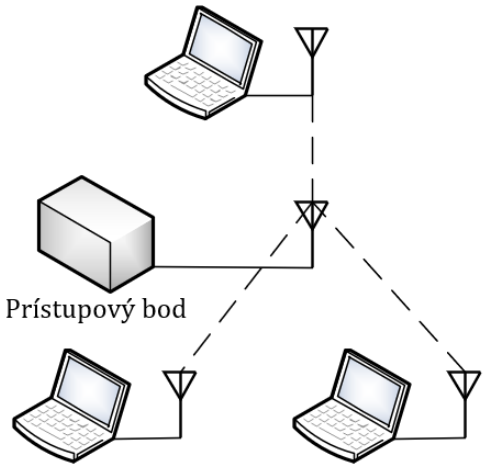
Bezdrôtové prenosové médium je prenosové médium, v ktorom sa šíri signál od jednej stanice k druhej. Pre štandard 802.11 sú najpoužívanejšie dve

prenosové rádiové frekvencie 2,4GHz, 5GHz a jedna infračervená. Rádiové frekvencie sú používané vo väčšine prípadov.

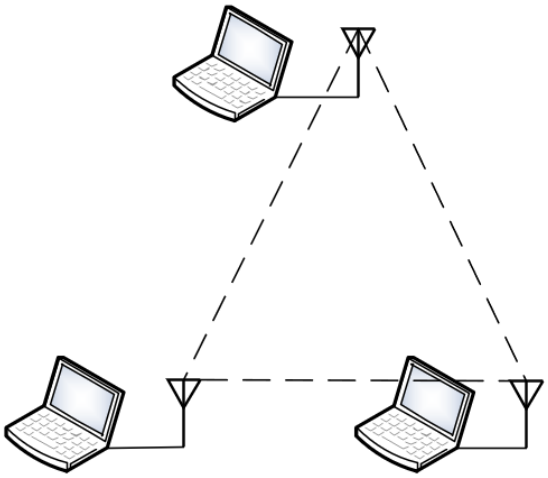
Distribučný systém je využívaný, ak sa vo veľkej oblasti nachádza množstvo prístupových bodov. Tieto prístupové body musia navzájom komunikovať, aby vedeli, kde sa konkrétna stanica nachádza. Špecifikácie distribučného systému nie sú pevne definované v štandarde. Zvyčajne sú prístupové body pospájané do určitej siete, ktorú nazývame chrbtová sieť. Väčšinou založenej na technológii Ethernet.

3.1.2 Architektúra WiFi sietí

Základný blok siete [29] podľa 802.11 sa nazýva BSS (basic service set). Je to skupina staníc, ktoré komunikujú medzi sebou. BSS môže mať niekoľko základných schém. Infraštruktúrne zapojene je zobrazené na obrázku Obr. 3.2 a ad-hoc zapojenie na obrázku Obr. 3.1. Ďalej sa používa zapojenie s rozšírenými servisnými oblasťami, zapojenie s virtuálnymi prístupovými bodmi (Multi BSS enviroments: „virtual AP“) a robustné bezpečnostné siete (Robust Security Networks).



Obr. 3.2 Infraštruktúrne zapojenie



Obr. 3.1 Zapojenie Ad-hoc

Infraštruktúrne zapojenie (Infrastructure BSS) [29] je zapojenie s prístupovým bodom. Prístupový bod sprostredkováva všetku komunikáciu medzi mobilnými uzlami v jednej servisnej oblasti. Znamená to, že ak chce jedno zariadenie

komunikovať s druhým v rovnakej servisnej oblasti, tak komunikácia bude predávaná prostredníctvom prístupového bodu. Pri sieťach s prístupovým bodom je dôležitým parametrom vzdialenosť mobilných staníc od prístupového bodu a nie vzájomná vzdialenosť mobilných staníc.

Stanice sa v tejto sieti musia asociovať s prístupovým bodom, aby bola možná ich komunikácia v sieti. Proces asociácie je logicky podobný ako zapojenie Ethernetového káblu do zariadenia. Mobilná stanica môže byť pripojená iba k jednému prístupovému bodu. Počet staníc, ktoré môže prístupový bod obsluhovať je limitovaný priepustnosťou siete.

Ad-hoc (Independent Networks) [29] zapojenie je zapojenie, pri ktorom stanice komunikujú priamo medzi sebou. Najmenšia Ad-hoc sieť má dve stanice. Väčšinou sú to malé siete vytvorené na krátky čas pre nejaký špeciálny účel ako je napríklad konferencia, počas ktorej si účastníci potrebujú vymieňať dáta.

Rozšírené servisné oblasti (Extended Service Areas) [29] sa používajú vo veľkých oblastiach. Prístupové body sú navzájom pospájané a tvoria chrbtovú sieť Wi-Fi siete a rozšírenú servisnú oblasť (Extended Service Set), ktorá je spojením BSS všetkých prístupových bodov tvoriacich chrbtovú sieť. Mobilná stanica sa vždy pripojí k prístupovému bodu s najväčším signálom.

Medzi [30] základné Wi-Fi štandardy patrí: Wi-Fi 802.11a – definuje formát a štruktúru rádiových signálov posielaných anténami a Wi-Fi smerovačmi. Využíva sa OFDM (Orthogonal Frequency-Division Multiplexing). Ďalej sa používajú štandardy Wi-Fi 802.11b, ktorý má priepustnosť 11Mb/s a pracuje na frekvenciách okolo 2.4 GHz. Používa sa hlavne v domácnostiach. Wi-Fi 802.11g prenáša dáta rýchlosťami do 54Mb/s a pracuje na frekvencii 2,4 GHz. Využíva OFDM. Wi-Fi [31] 802.11n dokáže prenášať dáta s rýchlosťami do 600Mb/s na frekvencii 2,4 GHz a 5GHz. Wi-Fi 802.11ac prenáša dáta na rýchlostiach do 1,3Gb/s pracuje na frekvencii 2,4 GHz aj 5 GHz. Wi-Fi 802.11ad dokáže prenášať dáta do rýchlostí 7Gb/s a vysiela na frekvencii 2,4 GHz, 5 GH/z a 60 GH/z.

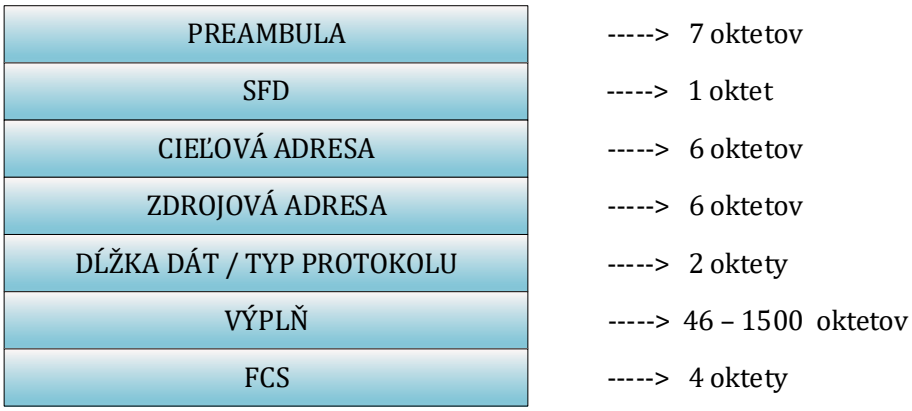
Wi-Fi siete sú poloduplexného charakteru. Obsahujú prístupový bod (Access Point), ktorý je prístupovým bodom k internetu. Prístupový bod vysiela rádiový signál do svojho okolia. Štandard IEEE 802.11 pre bezdrôtové LAN siete používa prístupovú technológiu CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance), pretože všetky stanice vysielajú a prijímajú na rovnakom rádiovom kanále. Problémom je, že rádiový kanál nemôže načúvať kým vysiela dáta, a potom nie je schopný detekovať kolíziu ako pri Ethernete. Práve z týchto dôvodov bola vyvinutá technológia DCF (Distribued Control Function) [32]. Vzhľadom k DCF Wi-Fi stanica vysiela iba ak je kanál voľný. Stanica posiela rámce s žiadosťou o pripojenie a čaká na potvrdenie, aby mohla začať vysielať. Ak jej potvrdenie nepríde, počká náhodný čas, a vyšle žiadosť o spojenie znovu. Čím viac zariadení sa k danému prístupovému bodu pripojí, tým nastane viac kolízií a rýchlosť sa zníži.

3.2 Ethernet

Ethernet [33] je najrozšírenejšia technológia používaná v sieťach LAN. Štandard ethernet definuje prepojovacie a signalizačné požiadavky pre fyzickú vrstvu vo vrstvovom modely TCP/IP. Pôvodne [33] bol Ethernet štandardizovaný pre rýchlosti do 10 Mb/s. Od vzniku štandardu Ethernet sa počítačové siete zmenili a rástli požiadavky na zvýšenie prenosových rýchlosti. Maximálne rýchlosti sa zvyšovali postupne na 100Mb/s, 1000Mb/s až po rýchlosti dosahujúce 10Gb/s. Prenosová rýchlosť 10Gb/s sa používa hlavne v dátových centrách. V súčasnosti sa testujú rýchlosti dosahujúce 100Gb/s. [34] Linky sú vo väčšine prípadov duplexné - umožňujú súčasný príjem aj vysielanie. Prenosové média pri Ethernete môžu byť koaxiálne káble napríklad 10Base2, UTP káble typu 10BaseT, 100BaseT, 1000BaseT alebo optické káble. Prenosové média sú spätne kompatibilné, avšak prenosová rýchlosť býva znížená podľa typu prenosového média. Ako konektory krútených párov sa používajú RJ-45.

3.2.1 Štruktúra rámca Ethernet

Na obrázku Obr. 3.3 je zobrazená štruktúra rámca Ethernet. [35] Ako prvá je preambula (Preamble) s dĺžkou 7 B, ktorá značí začiatok rámca a zabezpečuje synchronizáciu spolu s polom SFD (Start Frame Delimiter), ktoré má veľkosť 1 B. Nasleduje pole s dĺžkou 6 B obsahujúce cieľovú adresu (Destination address) a pole s rovnakou dĺžkou 6 B obsahujúce zdrojovú adresu (Source address). Pole dĺžka (Length) má veľkosť 2 B a obsahuje informácie o dĺžke rámca - počte platných bajtov v tele alebo identifikuje prenášaný protokol v tele rámca. Záťaž (Payload) môže byť od 46 B po 1500 B. Posledné pole obsahuje kontrolný súčet FCS (Frame Check Sequence) s veľkosťou 4 B. Kontrolný súčet býva vypočítaný pomocou cyklických kódov. Ak sa kontrolný súčet prijatého rámca zhoduje s vypočítaným kontrolným súčtom, rámec je prijatý ak sa nezhodujú rámec je ignorovaný.



Obr. 3.3 Štruktúra rámca Ethernet

3.2.2 Riešenie kolízií v zbernicovej topológii

Ethernet je definovaný v štandarde IEEE 802.3. Stanice prepojené technológiou Ethernet môžu byť fyzicky prepojené v zbernicovej topológii alebo hviezdicovej topológii. Avšak logická topológia je vždy zbernicová. Znamená to, že prenosové médium (kanál) je zdieľaný viacerými stanicami a iba jedna stanica ho v určitom čase môže používať. Všetky stanice v Ethernete príjmu rámec, ktorý si ponechá iba cieľová stanica, pre ktorú bol určený, ostatné stanice daný rámec zahodia. Cieľom je zabrániť stratám dát [36] kvôli kolíziám, ktoré sa vopred detekujú. Na zabránenie

kolíziám a strate dát sa používa technika CSMA/CD (Carrier sense multiple access with collision detection). Zariadenie zisťuje, či sa dáta prenášajú alebo nie. Ak žiadna stanica nevysiela dáta prostredníctvom prenosového média, tak čakajúca stanica začne vysielať a sleduje, či bol prenos úspešný alebo nie. Ak nastala kolízia začne stanica vysielať rámec znovu. Pri technológii Ethenet sa musí rátať s určitými obmedzeniami. Jednou z najväčších limitácií technológie Ethernet je dĺžka vedenia. Čím je dĺžka vedenia väčšia, tým je signál po ceste viacej zarušený čo môže spôsobiť chyby pri prenose a tiež oneskorenie signálu. Ďalší problém vzniká, ak dve stanice zistia, že aktuálne žiadna zo staníc nevysiela a začnú vysielať súčasne. Následne zaznamenajú kolíziu, počkajú náhodný čas a začnú vysielať znovu. Veľkosť rámcu pri použití technológie CSMA/CD je obmedzená.

Predtým, [36] ako sa pošle posledný bit rámca musí stanica, ktorá tento rámec vyslala zistiť, či došlo ku kolízii alebo nie. Dôvodom je, že ak je rámec raz prenesený v celej dĺžke, tak sa neudržuje jeho kópia. Vysielací čas Tfr musí byť minimálne dvakrát väčší ako čas šírenia rámcu k druhej stanici Tp. Táto vlastnosť je dôležitá pri detekcii kolízie. Ak sa dve stanice, ktoré sú v kolízii od seba nachádzajú v maximálnej vzdialenosti, tak signál z prvej stanice dosiahne druhú stanicu za čas Tp a trvá mu opäť čas Tp aby dosiahol prvú stanicu. Požiadavkou je, že prvá stanica môže vysielať po čase 2Tp.

3.3 Smerovacie protokoly

Smerovacie protokoly slúžia na tvorbu smerovacích tabuliek a umožňujú komunikáciu v sieti. Vo veľkých sieťach ako je Internet [36] musia pakety prejsť mnohými smerovačmi k dosiahnutiu cieľovej destinácie. Smerovač je zariadenie, ktoré príjma pakety a preposiela ich ďalej k cieľovej destinácii. Smerovač určuje cestu, ktorou paket pošle na základe metriky. Nižšia metrika znamená výhodnejšiu trasu. Jednotlivé cesty s východzími bránami (gateway) sú uložené v smerovacích tabuľkách. Smerovacie tabuľky môžu byť vytvorené staticky, konfigurované sú manuálne, čo je pri väčších sieťach alebo výpadkoch liniek problematické alebo sú vytvorené dynamicky. O dynamickú tvorbu smerovacích tabuliek sa starajú

smerovacie protokoly. Reagujú na výpadky liniek, zmenu stavu liniek a aktualizujú smerovacie tabuľky vzhľadom k aktuálnej situácii v sieti.

3.3.1 Základné rozdelenie

Smerovacie protokoly [37] majú viacero delení. Prvým je delenie na IGP (Interior Gateway Protocol) a EGP (Exterior Gateway Protocol). IGP protokoly sa používajú na prenos smerovacích informácií vo vnútri autonómneho systému. Partia sem protokoly RIP, RIPv2, EIGRP, OSPF, IS-IS. EGP protokoly sa používajú na komunikáciu medzi autonómnymi systémami. Patrí sem napríklad protokol BGP, ktorý je základným protokolom používaným v sieti Internet. Podľa spôsobu smerovania sa delia na link-state (protokoly zohľadňujúce stav linky) a distance-vector (protokoly zohľadňujúce vektor vzdialenosti).

Typickým protokolom skupiny **link-state** je protokol OSPF. Link-state protokoly [36] zisťujú informácie o svojich susedných smerovačoch. Smerovače pravidelne posielajú HELLO pakety a zisťujú vzájomnú dostupnosť. Následne smerovač posiela na skupinovú adresu (multicast) informácie o svojich susedoch. Každý smerovač v sieti postupne získa informácie o celej sieti a po prijatí paketu je schopný tento paket preposlať najvýhodnejšou trasou vypočítanou algoritmom implementovaným v protokole.

Sú náročnejšie na prostriedky CPU a tiež konfiguráciu. Z dôvodu zníženia zaplavovania siete réžiou smerovacieho protokolu sa smerovače rozdeľujú do oblastí - autonómnych systémov. Každý smerovač má tabuľku susedov, ktorá uchováva informácie o susedoch s rovnakým smerovacím protokolom, tabuľku topológie obsahujúcu informácie o topológii a smerovaciu tabuľku, ktorá obsahuje najlepšie cesty pre smerovanie paketov.

RIP a EIGRP sú najpoužívanejšími **distance-vector** protokolmi. Distance-vector protokoly [36] na rozdiel od link-state protokolov nemajú informácie o celej sieti. Poznajú sieť iba po najbližšieho suseda. Každá cieľová adresa má priradenú metriku. Nevýhodou týchto protokolov je dlhšie napĺňanie

a aktualizovanie smerovacích tabuliek oproti link-state protokolom. Metrikou je tu počet preskokov, smerovačov, v ceste. Ako algoritmus sa používa Bellman-Fordov algoritmus. Distance-vector protokoly sú jednoduchšie na konfiguráciu a využívajú väčšiu šírku pásma, pretože posielajú celé smerovacie tabuľky susedom. V pravidelných časových intervaloch zisťujú informácie o cestách. Ak nastane výpadok trasy, smerovač musí poslať celé aktualizované smerovacie tabuľky svojím susedom. Distance-vector protokoly používajú fixnú dĺžku masky, čo zapríčiňuje horšiu škálovateľnosť.

3.4 OSPF (Open Shortest Path First)

3.4.1 Popis protokolu OSPF

Protokol patrí do skupiny link-state [37] protokolov. OSPF smerovače posielajú všetkým priamo pripojeným susedom informácie o stave liniek. Každý smerovač, na ktorom sa aktivuje protokol OSPF posiela hello pakety všetkým priamo pripojeným smerovačom. Hello paket [38] obsahuje informácie o časovačoch a ID smerovača. ID smerovača je najvyššia adresa na Loopback, ak ho nemá smerovač nakonfigurovaný použije sa najvyššia adresa z ľubovoľného rozhrania. Router ID sa volí pri zapnutí smerovača. Hello pakety sa štandardne posielajú každých 10 sekúnd.

Ak sa parametre, ktoré si vymenili smerovače zhodujú, tak sa stanú OSPF susedmi. Naviaže sa spojenie a smerovače si vymenia databázu o stavoch liniek. Smerovače pripojené point-to-point alebo point-to-multipoint zostavujú spojenie automaticky. Posielajú si pakety, ktoré obsahujú informácie LSA (Link State Advertisement). Tieto informácie popisujú stav rozhrania a zoznam pripojených smerovačov. Smerovače LSA ukladajú do svojej lokálnej databáze a preposielajú ich na ostatné priľahlé smerovače. Tieto pakety majú TTL (Time To Live) s hodnotou 1.

Takýmto spôsobom sa informácie rozšíria do celej siete. Po získaní všetkých informácií každý smerovač pomocou Dijkstrovho algoritmu nájde najkratšie cesty do všetkých známych sietí a odstráni smyčky. Na základe týchto informácií sú naplnené smerovacie tabuľky. Ak dôjde k zmene topológie, smerovač na ktorom

došlo k zmene pošle priľahlým smerovačom informáciu LSA, na základe ktorej sa prepočítajú trasy na všetkých smerovačoch.

Každý smerovač posudzuje výhodnosť všetkých ciest. OSPF protokol [36] používa metriku s názvom cena (cost). Metrika nadobúda hodnoty v rozsahu 1 až 65536. Metrika sa vzťahuje ku konkrétnemu rozhraniu. Čím je menšia cena, tým je trasa výhodnejšia. Cena od zdrojovej adresy po cieľovú je sumou všetkých metrík po ceste.

3.4.2 Delenie na oblasti

Ak má smerovač [37] rozhranie konfigurované pre technológiu Ethernet, Frame Relay alebo FDDI, tak komunikáciu riadi referenčný smerovač (Designated Router) alebo záložný referenčný smerovač (Backup Designated Router), aby nebolo nutné nadväzovať spojenie „každý s každým“ čo by pri výmene LSU (Link State Update) významne zaťažovalo sieť.

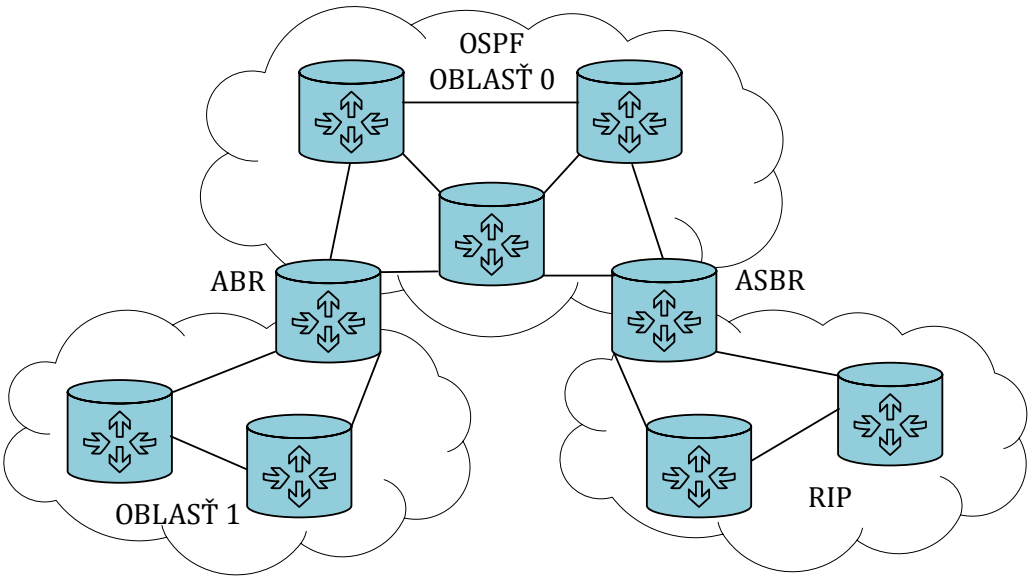
Protokol OSPF používa delenie na oblasti - rozdeľuje sieť na menšie celky. Je to logická skupina smerovačov a liniek medzi nimi. Každá oblasť obsahuje minimálne jeden až n smerovačov. Oblasti sú identifikované 32 bitovým číslom, ktoré sa zapisuje v desiatkovej sústave. V protokole OSPF je oblasť 0 (area 0) definovaná ako základná tranzitná úroveň (backbone). Smerovače v oblasti 0 sú vo väčšine prípadov spojené „každý s každým“ - full mesh topológia, čo zvyšuje spoľahlivosť a zabraňuje znefunkčneniu celej siete pri páde jednej z liniek.

Pri smerovačoch [38] patriacich do iných oblastí ako je oblasť 0 sa vyžaduje priame susedstvo oblasti so smerovačom patriacim do tranzitnej oblasti 0. Problém s touto podmienkou vzniká pri dodatočných zmenách siete. Aby sme novo pridanú sieť pripojili priamo k oblasti 0 používame virtuálnu linku. Virtuálna linka zabezpečuje logické spojenie medzi ABR (Area Border Router) - smerovač na rozhraní oblasti 0 a ABR smerovača na rozhraní novo pridanej siete. Virtuálne linky nesmú prechádzať cez viac ako jednu oblasť. Používajú sa v nevyhnutných prípadoch. Členenie na oblasti je významné pri veľkých sieťach, redukuje záznamy v smerovacích tabuľkách, čím znižuje počet poslaných LSU.

Smerovač môže byť v dvoch oblastiach naraz, pretože oblasť sa definuje na rozhraní. Smerovače delíme na vnútorné smerovače, hraničné smerovače oblastí a hraničné smerovače autonómnych systémov. Vnútorné smerovače sú také, že každé rozhranie má pridelenú rovnakú oblasť. Hraničné smerovače oblastí (Area Border Router) majú rozhrania, na ktorých je aktivovaný protokol OSPF pridelené do rôznych oblastí pričom jedna je tranzitná. Hraničné smerovače autonómnych systémov (Autonomous System Border Router) sú smerovače, ktoré spájajú autonómne systémy navzájom. Znamená to, že môžu spájať aj siete s odlišným smerovacím protokolom ako je zobrazené na obrázku Obr. 3.4.

3.4.3 Redistribúcia

Pri [38] používaní rôznych protokolov v sieti sa musí byť použitá redistribúcia. Znamená to, že na jednom smerovači bude napríklad aktivovaný protokol OSPF aj EIGRP v jednom čase. Takýto smerovač označujeme hraničným smerovačom autonómneho systému (Autonomous System Boundary Router, ASBR). Pri redistribúcii sa musí nakonfigurovať cena cesty z EIGRP alebo RIP do OSPF, pretože každý protokol má vlastnú interpretáciu metriky a nebolo by možné vyhľadať najvýhodnejšiu trasu.



Obr. 3.4 Rozdelenie na oblasti a redistribúcia protokolu OSPF

Protokol OSPF [38] podľa toku dát prostredníctvom oblastí delíme na OSPF s tranzitnou oblasťou, stub oblasťou alebo totally stubby area (rozšírenie pre Cisco zariadenia). Ak je oblasť nakonfigurovaná ako tranzitná, znamená to, že dáta môžu byť smerované cez túto oblasť. Ak je oblasť stub oblasťou, dáta nemôžu touto oblasťou prechádzať. Vždy v nej musia končiť alebo začínať. Pri stub oblasti sa do externých sietí smeruje pomocou defaultnej cesty, ktorá je propagovaná zo smerovača ASBR. Nie sú do nej propagované externé cesty, ale len cesty z daného autonómneho systému. Stub oblasti sa používajú na zmenšenie smerovacích tabuliek. Najčastejšie sa stub oblasti definujú v prípadoch, ak z nich vedie iba jedna cesta von. Totally stubby area má v smerovacích tabuľkách definovanú iba defaultnú cestu a cesty vo vnútri oblasti.

3.4.4 Sumarizácia

IP adresy pri protokole OSPF [38] môžeme sumarizovať. Sumarizovať znamená priradiť niekoľko IP adries s určitou maskou siete do jednej IP adresy so zmenenou maskou siete. Aby bolo možné IP adresy sumarizovať, musia spĺňať určité podmienky. Musia nasledovať za sebou a byť mocninou dvojky a v inej časti siete neexistovala žiadna podsieť, ktorá by do tejto sumárnej cesty spadala. Znamená to, že sumarizovať môžeme 2, 4, 16 adries. Sumarizácia zabraňuje zbytočnému posielaniu všetkých podsietí medzi oblasťami. Namiesto všetkých podsietí sa pošle len sumárna cesta. OSPF podporuje variabilnú dĺžku masky na rozdiel od protokolu RIP, kde musí mať maska pevnú dĺžku.

4 PRAKTICKÁ ČASŤ

Cieľom diplomovej práce je vypracovať laboratórne úlohy pre výuku sieťových technológií. Laboratórne úlohy majú byť realizované vo virtuálnom prostredí pomocou vhodného simulačného prostredia. Základnými požiadavkami je, aby dané simulačné prostredie nekolidovalo inými vyučovanými predmetmi. To znamená, že by sa nemalo jednať o konfiguráciu zariadení CISCO ani MikroTik. Vybrané simulačné prostredie musí byť bezplatné (free verzia), alebo vo verzii s otvoreným kódom (opensource). Tiež je nutné aby malo podporu vývojárov, keďže má slúžiť ako náhrada simulačného prostredia RiverBed Academic Edition 17.5, ktorému podpora končí.

4.1 Výber simulačného prostredia pre vypracovanie práce

Pri výbere vhodného simulačného prostredia bolo nutné zohľadňovať určité požiadavky. V simulačnom prostredí použitom na tvorbu laboratórnych úloh musí byť možné zobrazovať smerovacie tabuľky, sledovať tok dát formou určitej grafickej simulácie, zachytávať pakety pomocou vhodného programu na sledovanie sieťového toku dát a vykresľovať grafy dátového toku, stratovosti paketov, zaťaženia liniek a iných veličín pre lepšie pochopenie preberanej látky. Na internete bolo nájdených viacero simulačných prostredí. Na základe ich funkcií, možností, vývojárskej podpory a vhodnosti použitia bolo nakoniec zvolené jedno konkrétne simulačné prostredie, v ktorom boli realizované laboratórne úlohy. Následne budú popísané niektoré z nájdených simulačných programov. Ich funkcie, možnosti, výhody, nevýhody.

4.1.1 CORE (Common Open Research Emulator)

Toto simulačné prostredie poskytuje grafické užívateľské rozhranie. Pracuje v systéme Linux. Používa Linuxové kontajnery (LXC) ako virtualizačnú technológiu. CORE vytvára unikátny menný priestor pre každý uzol. Tým pádom má každý uzol vlastné funkcie, smerovacie tabuľky a IP adresy. Cieľom je použiť čo najmenšie

prostriedky pri vytváraní každého uzlu a tým znižovať systémové požiadavky. Je v ňom možné vytvárať množstvo virtuálnych strojov relatívne rýchlo. Simuluje reálnu počítačovú sieť, ktorá beží v reálnom čase. CORE [39] rozširuje prostredie IMMUNES. Používa knižnice Pythonu. Podporuje tiež bezdrôtové siete. Grafické užívateľské rozhranie je rozdelené do dvoch vrstiev. Na zariadeniach druhej vrstvy L2 sa nachádzajú rozbočovače, prepínače, bezdrôtové LAN emulátory. Tretia vrstva L3 zahŕňa koncové stanice a smerovače. Zariadenia tretej vrstvy môžu používať preddefinované služby ako aj služby definované užívateľom. Prípadne sa dajú upraviť rôzne protokoly podľa potrieb. Prostredie CORE má možnosť zobrazenia smerovacích tabuliek a tiež je možné sledovať tok dát na jednotlivých rozhraniach pomocou programu Wireshark.

4.1.2 Cloonix

Je to IP simulátor, ktorý poskytuje prehľadné grafické užívateľské prostredie. Používa QEMU/KVM [39] na vytvorenie virtuálnych zariadení KVM (Kernel-based Virtual Machine). Je to virtualizačná technika, pri ktorej sa Linuxové jadro zmení na hypervízora a dovolí na jednom počítači spustiť viacero operačných systémov. Virtualizované operačné systémy sa nazývajú hostia. QEMU býva používané v kombinácii s KVM na dosiahnutie vysokých rýchlostí virtuálnych strojov, ktoré sa približujú reálnym rýchlostiam. Je to možné kvôli dynamickému binárnemu prekladu pre kód jadra operačného systému. Cloonix používa širokú paletu predpripravených systémov súborov, ktoré môžu byť použité ako virtuálne stroje a poskytujú jednoduché inštrukcie na vytvorenie ďalších systémov súborov. Tiež je možné sledovať tok dát pomocou programu Wireshark. Grafy v simulačnom prostredí Cloonix nie je možné zobrazovať.

4.1.3 GNS3

Je simulačné prostredie s grafickým užívateľským rozhraním. GNS3 je primárne zamerané pre zariadenia CISCO a Juniper a používané prevažne k príprave k certifikácii CISCO. Simulačné prostredie poskytuje širokú škálu protokolov, veľa príkladov na rôzne topológie a prepracovanú dokumentáciu s oficiálnymi

tutoriálmi. GNS3 je možné nainštalovať na operačný systém Windows aj na operačný systém Linux, čo je výhodou. GNS3 je v podstate virtuálny operačný systém CISCO. Topológia sa vytvára veľmi jednoducho štýlom „ťahaj a pusti“ (drag and drop). Konfigurácia je realizovaná pomocou terminálov jednotlivých zariadení. Podporované je veľké množstvo príkazov. Pre niektoré príkazy daná konfigurácia nie je možná bez fyzického zariadenia. V prostredí GNS3 sa dajú sledovať smerovacie tabuľky aj aktuálna konfigurácia a tok paketov avšak tento program nie je vhodný vzhľadom k stanoveným požiadavkám. Priepustnosť paketov sa blíži k hodnote 1000 paketov/s. Reálny smerovač má priepustnosť niekedy tisíckrát vyššiu.

4.1.4 Mininet

Používa Linuxový [39] sieťový menný priestor a Linuxovú technológiu na tvorbu virtuálnych uzlov siete. Bol vytvorený pre potreby výskumu softvérovo definovaných sieťových technológií. V simulačnom prostredí Mininet je možné vytvoriť tisícky virtuálnych uzlov. Avšak Mininet je vhodný hlavne na tvorbu SDN (Software-defined networking) sietí. Grafické užívateľské rozhranie poskytuje len málo možností a je potrebná konfigurácia pomocou príkazového riadku, čím sa naskytuje viacej možností použitia avšak je vyžadovaná znalosť programovacieho jazyka Python. Dokumentácia je kvalitne spracovaná a Mininet je používaný širokou komunitou užívateľov.

4.1.5 NetSim Academic

Je simulačné prostredie pre laboratórne a výukové účely. [40] Má prehľadné grafické užívateľské rozhranie. Podporuje mnohé smerovacie protokoly a technológie, štandardy 802.11, 802.15.4, LTE siete, IOT (Internet of Things) technológie, WiMax, Ethernet, Aloha, GSM & CDMA, TCP a mnohé ďalšie. NetSim má prepracovanú dokumentáciu aj oficiálne tutoriály. Inštalácia je jednoduchá pre všetky platformy. Taktiež je zahrnutá podpora od vývojárov tohto softvéru. Zo simulačného prostredia NetSim je ako výstup možné získať grafy a smerovacie tabuľky a tiež je možné sledovať tok paketov idúcich cez sieť pre lepšie pochopenie

úlohy. Ako nakoniec bolo zistené, NetSim Academic nie je bezplatný ani pre výukové účely, preto nespĺňa stanovené podmienky na jeho použitie.

4.1.6 NS-3

Je diskrétny simulačný program [41] pre sieťové služby a simulovanie sieťového toku a jeho analyzovanie. Využíva sa primárne pre výskum a vzdelávacie účely. NS-3 je softvér, v ktorom sa vytvárajú simulácie prostredníctvom kódu v programovacom jazyku C++. Pri jeho použití sú potrebné predošlé skúsenosti s programovaním.

NS-3 je náhradou za NS-2, ktorá je však stále používaná avšak nie je spätne kompatibilný. NS-3 je úplne nový program. Je to simulačný softvér s otvoreným kódom. Projekt NS-3 má prepracovanú dokumentáciu a podporu vývojárov. Inštalácia ja možná na operačný systému Linux. Používa sa programovací jazyk C++ a Python. Podporuje IP siete všetkých typov. Patrí sem napríklad Wi-Fi, LTE (Long Term Evolution) a iné. Dajú sa v nej simulovať smerovacie protokoly OSPF (Open Shortest Path First), RIP (Routing Information Protocol), transportné protokoly TCP (Transmission Control Protocol), UDP (User Datagram Protocol), smerovať pakety s IPv4 aj IPv6 adresami a veľa ďalších protokolov a technológií.

Výhodou je, že môžeme sledovať smerovacie tabuľky a zachytávať tok paketov programom Wireshark. Tiež je tu možnosť vykresliť grafy zaťaženia liniek, oneskorenia, jitteru, zahadzovania paketov pomocou nástroja Gnuplot a možnosť sledovania toku paketov v reálnom čase pomocou nástroja NetAnim. Nevýhodou je skutočnosť popisovaná vyššie a to, že NS-3 nemá grafické užívateľské rozhranie a je nutné používať programovací jazyk C++. Na vytváranie topológií a simulácií v C++ sa používajú rôzne vývojové prostredia–IDE (Integrated development enviroment). Najpoužívanejšími pre simulátor NS-3 sú Eclipse, NetBeans a QtCreator.

Existuje aj rozšírenie DCE (Direct Code Execution) s priamym vykonávaním kódu a rozšírenie Quagga, ktoré podporuje protokoly ako BGP, BGP+, OSPFv3, RIPv2, RIPng a ďalšie a má možnosť realizovať konfiguráciu pomocou príkazového riadku alebo písaním kódu v C++. Je voľne distribuovaný pod licenciou GPL.

4.1.7 Psimulator2

Je to [42] simulátor, ktorý disponuje grafickým užívateľským rozhraním. Vytvorený bol v Prahe na Fakulte Informačných Technológii ČVUT pre výukové účely. Je možné spustiť ho na akomkoľvek operačnom systéme, ktorý podporuje Javu. Ku každému pridanému zariadeniu je možné pristupovať pomocou protokolu TELNET a následne ho konfigurovať v príkazovom riadku. Užívateľ si vytvorí topológiu v grafickom užívateľskom rozhraní štýlom „ťahaj a pusti" následne ju nakonfiguruje. Simulácia beží v pozadí na základe informácii v XML súbore, v ktorom sú informácie o vytvorenej topológii. Simulátor umožňuje zobrazovanie poslaných prípadne zahodených paketov.

4.2 Zvolené simulačné prostredie a IDE

Na základe zistených výhod, nevýhod, možností dostupných simulačných prostredí s otvoreným kódom (open-source) a požiadavkách bolo zvolené prostredie NS-3. Ako bolo spomenuté v kapitole 4.1, toto prostredie podporuje množstvo technológií, umožňuje zachytávať pakety pomocou programu Wireshark, vykresľovať grafy a animovať výslednú topológiu simuláciou toku dát k jednotlivým uzlom a koncovým staniciam. Všetky príklady z dokumentácie a tutoriálu sú pod licenciou GPL, sú slobodne šíriteľné a upravovateľné.

Pre laboratórne úlohy bolo zvolené vývojové prostredie Eclipse. S daným vývojovým prostredím sa študenti stretli na viacerých predmetoch v rámci oboru a museli v ňom vypracovať viaceré projekty a úlohy. Preto by nemali byť komplikácie s použitím z daného vývojového prostredia pri riešení a študent sa môže zamerať priamo na konkrétne riešenie laboratórnej úlohy.

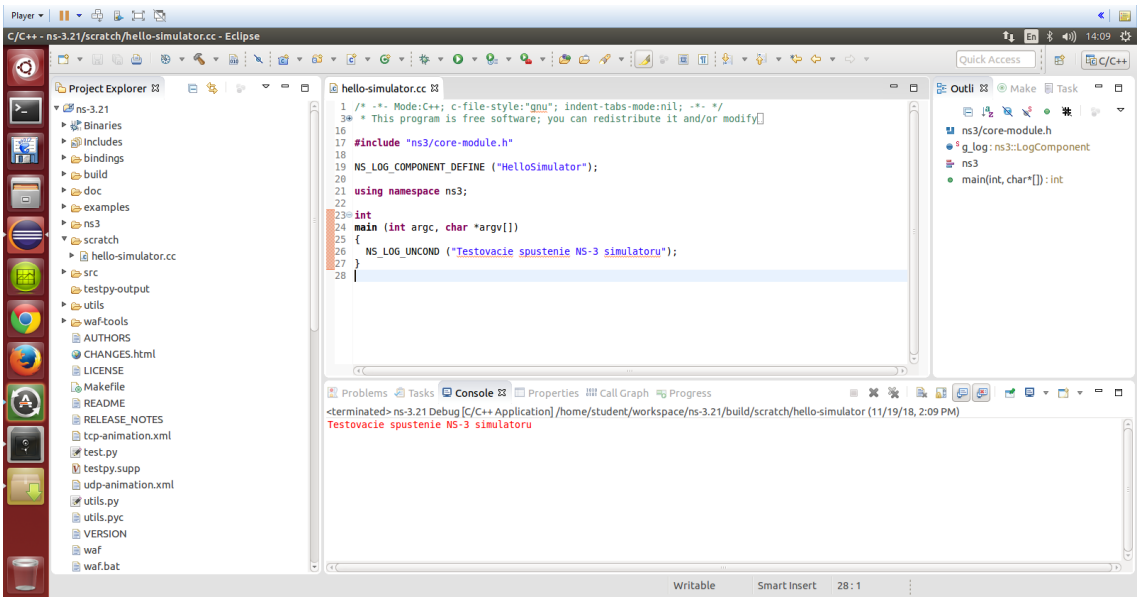
V tomto simulačnom prostredí sa využívajú `Topology Helpery` [41], ktorými sa práca zjednodušuje. Napríklad automaticky priraďujú MAC adresy, definujú prenos cez fyzické rozhranie. Zabezpečujú, aby základné úkony boli čo najjednoduchšie. Informácie o topology helperoch sa dajú nájsť v dokumentácii.

Eclipse je vývojové prostredie s otvoreným kódom. Je založené na pluginoch. To znamená, že je veľmi flexibilné. Pomocou pluginov môžeme toto prostredie

rozšíriť o množstvo programovacích jazykov Java, C++, C, Python, LaTex a iných. Podporuje [43] rôzne iné funkcie napríklad tvorbu GUI (Graphic User Interface)-grafického užívateľského rozhrania, funkciu generovania UML (Unified Modeling Language) diagramov, SysML (System Modeling Language), OCL (Object Constaint Language) zapisovania HTML, XML.

Podporuje skoro všetky operačné systémy a rôzne techniky analýzy kódu. Je veľmi robustné. Zaujímavá je aj podpora Androidu ADT (Android Development Tools). Tento plugin slúži na tvorbu Android aplikácií. ADT rozširuje vývojové prostredie Eclipse o možnosti tvorby Android projektov použitím Android SDK (Software Development Kit) nástrojov. Je to jedno s najpoužívanejších vývojových prostredí súčasnosti. Eclipse má mnohé ďalšie funkcie a možnosti, ktoré nebudú ďalej popisované.

Sieťový simulátor NS-3 bol nainštalovaný na operačný systém Ubuntu, ktorý bol virtualizovaný pomocou WMware Workstation 14 Player nainštalovanom na operačnom systéme Windows 10 64bit. Následne bol nainštalovaný Eclipse s IDE pre C/C++, doinštalované balíčky a vykonané nastavenia potrebné pre správny beh simulátoru NS-3. Ukážka zvoleného vývojového prostredia pre tvorbu modelov s testovacím výpisom je zobrazená na obrázku Obr. 4.1.



Obr. 4.1 Ukážka vývojového prostredia pre prácu s NS-3

5 NÁVRH LABORATÓRNYCH ÚLOH

V tejto kapitole sú navrhnuté dve laboratórne úlohy. Prvá sa zaoberá problematikou technológií WiFi a Ethernet. Druhá je zameraná na smerovací protokol OSPF. K laboratórnym úlohám sú vytvorené návody s postupom práce, samostatnými úlohami, očakávanými výsledkami a kontrolnými otázkami.

5.1 Založenie projektu pre prácu so simulátorom NS-3

Aby bolo možné vytvárať konkrétnu topológiu siete pomocou sieťového simulátora NS-3 implementovaného do vývojového prostredia Eclipse a následne vypracovávať laboratórne úlohy, tak je nutné dodržať určitý postup práce popísaný ďalej. Projekt má príponu `*.cc` – je písaný v programovacom jazyku C++. Aby bola možná jeho kompilácia, musí byť tento súbor `*.cc` umiestnený v zložke `scratch`. V tejto zložke nesmie byť naraz viac ako jeden projekt.

Zložka `scratch` sa nachádza v `/home/nazov_uzivatela/ns-allinone-3.21/ns-3.21`. Pôvodný súbor, „hello-simulator.cc“, ktorý sa v zložke `scratch` nachádza je potrebné zmazať. Následne je potrebné v okne Project Explorer rozbaliť zložku `ns-3.21` a prejsť do zložky `scratch` pravým tlačidlom otvoriť možnosti a kliknúť na `Refresh`. Zložka bude prázdna.

Kliknutím na zložku `scratch` pravým tlačidlom bude zvolené `New→File`. Názov bude zvolený napríklad „`Wi-Fi.cc`“. Dôležité je uviesť aj príponu, inak projekt nebude možné skompilovať. Keďže musí byť projekt skompilovaný, je nutné, aby bol vytvorený jednoduchý funkčný program, ktorý vypíše do konzole textový reťazec. Na overenie funkčnosti simulátoru a prvé spustenie kompilátoru je potrebné vložiť nasledujúci kód do novo vytvoreného súboru „`Wi-Fi.cc`“ .

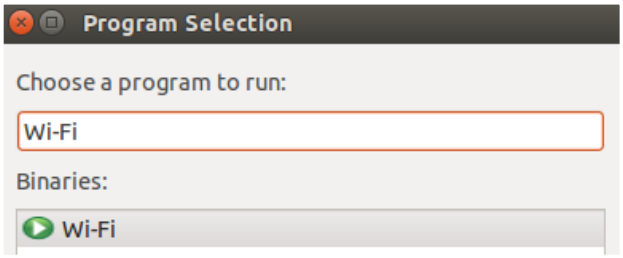
```
#include "ns3/core-module.h"
NS_LOG_COMPONENT_DEFINE ("WiFi");
using namespace ns3;

int main (int argc, char *argv[])
{
     NS_LOG_UNCOND ("Testovaci textovy retazec");
}
```

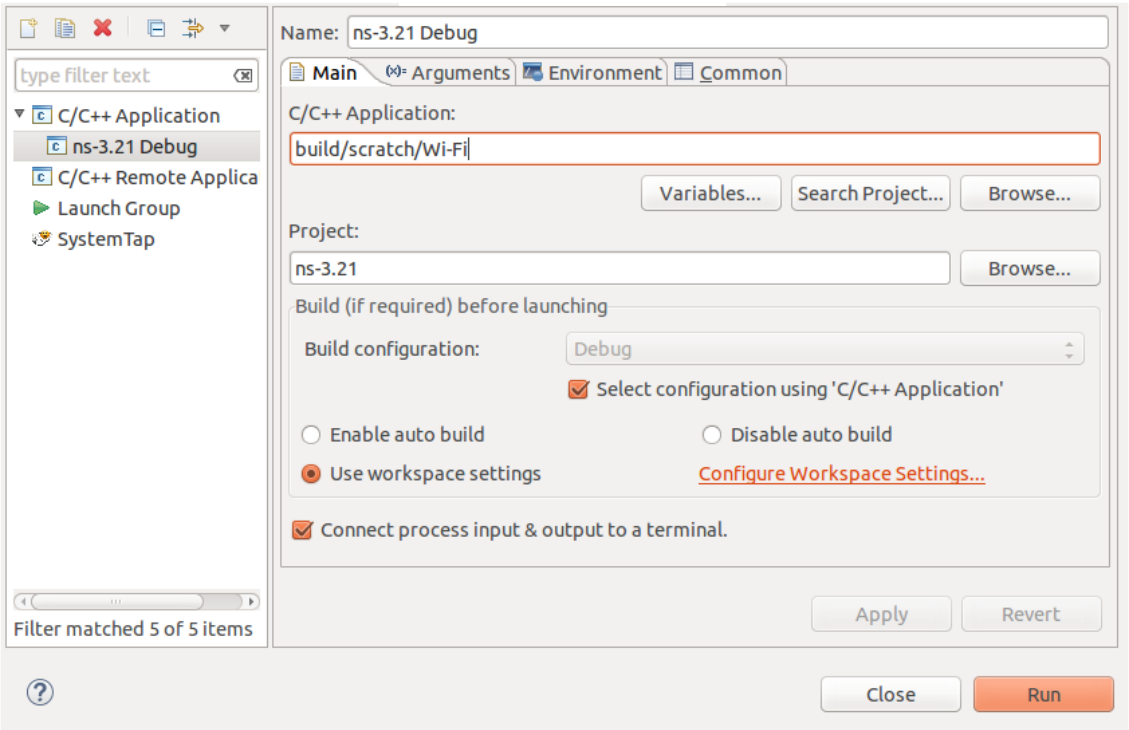
Ďalším krokom je spustenie kompilátoru. Spúšťa sa z `Menu→Run` (zelená šípka). Týmto sa vytvoria pomocné súbory potrebné pre debugger. Skompilované budú všetky projekty, ktoré sú v NS-3 simulátore vytvorené. Kompilácia bude trvať niekoľko sekúnd. V konzoly sa aj napriek zmene v programe objaví pôvodná hláška „Hello simulator“.

Aby sa v konzole zobrazil výpis z novo vytvoreného programu, tak je potrebné prejsť do nastavení cesty pre debugger. Nastavenie sa realizuje výberom položky `Run` (Zelená šípka)`→Run Configurations...`

Po otvorení dialógového okna po kliknutí na `Search project...`, musí byť vpísaný názov vytvoreného projektu (v tomto prípade „Wi-Fi“), vybraný projekt sa potvrdzuje tlačidlom „Ok“. Skompilovaný projekt musí mať vedľa názvu zelenú šípku Obr. 5.1. Výsledná konfigurácia je na Obr. 5.2.

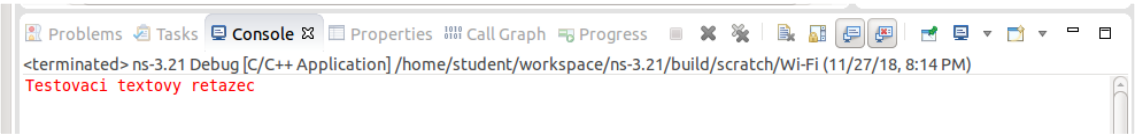


Obr. 5.1 Výber projektu



Obr. 5.2 Ukážka správneho nastavenia projektu

Po nakonfigurovaní a spustení tlačidlom Run sa vykonajú inštrukcie z programu „`Wi-Fi`“, do konzoly sa vypíše hláška „Testovaci textovy retazec“. Výsledok je zobrazený na obrázku Obr. 5.3.



Obr. 5.3 Ukážka výpisu z konzole

Ak sa v konzoly zobrazí hláška z novo vytvoreného projektu znamená to, že je všetko nastavené správne a môže sa pristúpiť k riešeniu konkrétnych úloh. Pred vypracovávaním úlohy je nutné vymazať celý kód v novo vytvorenom, skompilovanom súbore a pokračovať ďalej podľa návodu.

5.2 Štruktúra kódu

Zdrojový kód musí mať určitú štruktúru a dodržiavať zásady. Prvou časťou sú definície hlavičkových súborov. Majú príponu `*.h` a zadávajú sa za slovo `#include`. Ich pridaním sa sprístupnia predpripravené moduly v novovytvorenom projekte, aby sa s nimi dalo pracovať. Podľa povahy úlohy sú postupne pridávané potrebné moduly. Základným hlavičkovým súborom je súbor `core-module.h`. Príklad pridania modulu v prostredí NS-3:

```
#include "ns3/core-module.h"
```

Pri vytváraní modelu je nutné sprístupnenie menného priestoru NS-3. Sprístupnenie sa vykonáva kódom „`using namespace ns3;`“.

```
using namespace ns3;
```

Aby sa získal prístup k logom a následne sa dali hľadať prípadné chyby v kóde musí sa logovanie aktivovať nasledujúcim kódom:

```
NS_LOG_COMPONENT_DEFINE ("Nazov");
```

Vlastný kód so všetkými funkciami sa píše do funkcie `int main(){}`.

```
int main (int argc, char *argv[])
{ }
```

Počas laboratórnej úlohy budú do tela kódu postupne vkladané časti kódu potrebné k vytvoreniu topológie, priraďovanie IP adries a celkovej konfigurácie.

Na konci funkcie `int main(){}` po definícií topológie je potrebné simulátor spustiť. Spustenie sa vykonáva pomocou funkcie `Simulator::Run()`. Táto funkcia je zodpovedná za správne načasovanie udalostí definovaných v kóde.

```
Simulator::Run ();
```

Po ukončení simulácie, po odoslaní všetkých paketov, sa zastavia všetky aplikácie. Aby sa uvoľnili alokované prostriedky je použitý nasledujúci kód:

```
Simulator::Destroy ();
return 0;
```

5.3 Základná štruktúra laboratórnych úloh

Navrhnuté laboratórne úlohy sú zamerané na sieťové technológie a majú rozšíriť znalosti nadobudnuté na prednáškach. Obe úlohy majú rovnakú štruktúru a sú zamerané každá na inú časť preberanej látky. Prvá časť pozostáva z teoretického úvodu, ktorá slúži ako úvod do danej problematiky. V ďalšej časti návodu je popísaný konkrétny postup riešenia laboratórnej úlohy s vysvetlením jednotlivých krokov a časťami kódu, ktoré sa budú počas riešenia laboratórnej úlohy písať do vývojového prostredia Eclipse, v ktorom je implementovaný simulátor NS-3. V ďalšej časti budú zobrazené výstupy z programu Wireshark, grafy, simulácie, smerovacie tabuľky v závislosti na možnostiach danej úlohy, aby si študent mohol skontrolovať, či postupuje správne, vrátiť sa k prípadným chybám a odstrániť ich. Následne bude zadaná samostatná práca, ktorú po vypracovaní predošlých krokov bude študent schopný splniť. Na konci laboratórnej úlohy je zadaných niekoľko kontrolných otázok.

5.4 Popis laboratórnej úlohy „Technológia Wi-Fi a Ethernet“

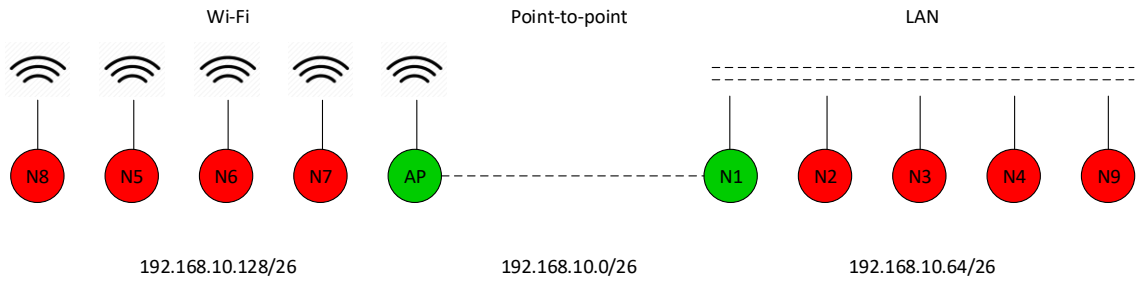
Laboratórna úloha sa zameriava na implementáciu bezdrôtovej technológie Wi-Fi a technológie Ethernet v simulačnom prostredí NS-3.

V laboratórnej úlohe je vytvorená topológia podľa obrázku Obr. 5.4, kde bezdrôtová stanica N8 s adresou 192.168.10.132/26 pošle požiadavku, `Request`, ECHO protokolu na stanicu N9 s adresou 192.168.10.69/26 komunikujúcu prostredníctvom technológie Ethernet v LAN topológii. Stanica odpovie správou odpoveď, `Reply`. Následne prebehne rovnaká komunikácia medzi uzlami N7, IP adresa 192.168.10.131/26 a N4, IP adresa 192.168.10.68/26. Štatistiky o priepustnosti, počte odoslaných, počte prijatých paketov, oneskorení a jitteri sú pre tieto dva toky vypísané do konzoly.

Vytvorená je aplikácia, ktorá posiela ping na uzol N3 (192.168.10.67/26) z uzlov N2, adresa 192.168.10.66/26, N5, adresa 192.168.10.130/26 a N6 adresa 192.168.10.129/26. Následne je meraný čas RTT (Round Trip Time) na základe čoho sa porovná čas RTT medzi dvoma uzlami v LAN sieti a RTT čas medzi uzlom z LAN siete a uzlom z Wi-Fi siete. Wi-Fi stanice sa pseudo-náhodne pohybujú v presne vymedzenom priestore. Súradnice pre ľubovoľný uzol môžu byť vypísané do konzoly a trajektória uzlov je zobrazená v programe NetAnim.

Vygenerované sú trasovacie súbory `*.pcap` pre prístupový bod a prvý uzol topológie LAN spojený s prístupovým bodom, ktoré je možné otvoriť v programe Wireshark, smerovacie tabuľky v textovom súbore a súbor `animace_WiFI.xml` pre vizualizáciu v programe NetAnim. Ako posledné je manuálne nastavovaná vzdialenosť uzlov od prístupového bodu pre štandardy 802.11a, 802.11b, 802.11g, 802.11n, 802.11ac, 802.11ad a zisťovaný možný dosah, kedy ešte stanica príjma signál od prístupového bodu v interiéri a exteriéri a kedy je vzdialenosť príliš veľká.

Laboratórna úloha je popisovaná tak, ako vyzerá po vypracovaní všetkých samostatných úloh. V prílohe PRÍLOHA A1 sa nachádza vypracovaný návod, ktorý obsahuje teoretický úvod, návod na vytvorenie základnej topológie, samostatné úlohy a kontrolné otázky.



Obr. 5.4 Konečná topológia laboratórnej úlohy Technológia Wi-Fi a Ethernet

5.5 Popis laboratórnej úlohy „Smerovací protokol OSPF“

Laboratórna úloha sa zaoberá implementáciou protokolu OSPF v simulačnom prostredí NS-3.

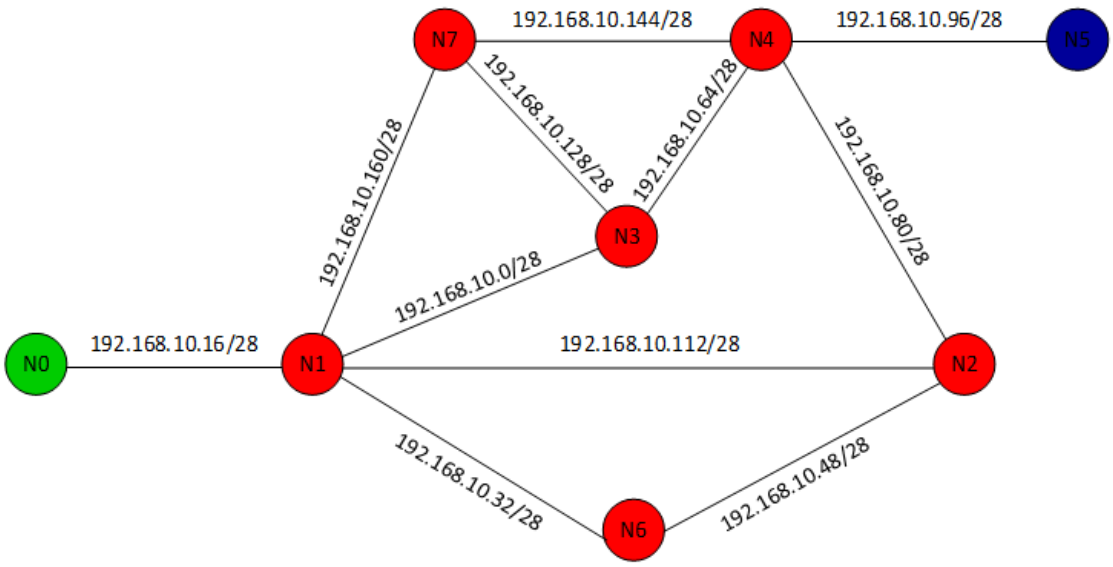
Postupne je vytvorená topológia siete, nastavené sú IP adresy podľa obrázku Obr. 5.5. IP adresy sú vyberané z rozsahu 192.168.10.0 s maskou /28 (255.255.255.240). Vytvorená je UDP a TCP aplikácia na generovanie toku paketov. UDP protokolom je prenášaná služba TFTP (Trivial Transfer File Protokol) na porte 69 a protokol TCP zabezpečuje službu HTTPS (Hypertext Transfer Protocol Secure) na porte 443.

Názorne je ukázané, ako sa správa protokol OSPF pri zmenách metriky liniek na rozhraniach. V ďalšom scenári je simulovaný výpadok niektorých rozhraní. Na základe zmien metriky a výpadkov liniek sú zaznamenávané zmeny v smerovaní paketov pomocou programu NetAnim a pozorované sú zmeny v smerovacích tabuľkách. Smerovacie tabuľky sú generované pred výpadkom určitých rozhraní a po výpadkoch niektorých rozhraní v časoch 2 s, 6 s a 9 s. Generované budú `*pcap` súbory pre každý uzol v topológii, ktoré bude možné otvoriť v sieťovom analyzátore Wireshark. Tieto súbory sú následne analyzované. Vytvorený je súbor `animace_OSPF.xml`, ktorý vizualizuje topológiu v programe NetAnim. V konzole sa zobrazuje počet odoslaných paketov, počet doručených paketov, oneskorenie, jitter a priepustnosť. Pri protokole UDP jednosmerne pri protokole TCP obojsmerne, čo súvisí so spojovou orientáciou TCP protokolu. Taktiež je zobrazený

graf prenosovej rýchlosti paketov medzi uzlom N3 a N4, kde je vidieť výpadok a presmerovanie toku na iné rozhranie.

Klient je nainštalovaný na uzol číslo 5 (N5) s adresou 192.168.10.98/28 a server je nainštalovaný na uzol 0 (N0) s adresou 192.168.10.17/28. Pakety sú posielané na rozhranie i0i1 k uzlu N0 najskôr prostredníctvom smerovačov N4, N3, N1 následne v čase 5 s prostredníctvom smerovačov N4, N2, N1 a nakoniec v čase 7 s prostredníctvom N4, N7, N1. Po zmene metriky bude smerovanie prebiehať prostredníctvom uzlov N4, N2, N6, N1.

Laboratórna úloha je popisovaná tak, ako vyzerá po vypracovaní všetkých samostatných úloh. V prílohe PRÍLOHA A2 sa nachádza vypracovaný návod, ktorý obsahuje teoretický úvod, návod na vytvorenie základnej topológie, samostatné úlohy a kontrolné otázky.



Obr. 5.5 Konečná topológia laboratórnej úlohy smerovací protokol OSPF

6 VYPRACOVANIE LABORATÓRNYCH ÚLOH A POPIS KÓDU S VÝSLEDKAMI

V tejto kapitole sú postupne riešené laboratórne úlohy a popisovaný význam blokov kódu, ktorý je kopírovaný do vývojového prostredia Eclipse, nainštalovaným vo virtuálnom stroji s operačným systémom Ubuntu so sieťovým diskrétnym simulátorom NS-3. Následne sú zobrazené výsledky a očakávané výstupy závisiace na konkrétnej laboratórnej úlohe. Laboratórne úlohy sú vytvorené pomocou dokumentácie NS-3 [41], NS-3 tutoriálu [44] a NS-3 doxygen [45].

6.1 Tvorba simulačného modelu pre laboratórnu úlohu Technológia Wi-Fi a Ethernet

Pred začiatkom laboratórnej úlohy musí byť vytvorená zložka `laboratorna_uloha` v `/home/ns-allinone-3.21/ns-3.21`. Následne bude skopírovaný obsah súboru `Wi-Fi.txt` do novovytvoreného projektu. Topológiu zobrazenú na obrázku Obr. 5.4 je nutné preniesť do skriptu v jazyku C++. Laboratórna úloha je vytvorená pomocou tutoriálu ns-3 [44] a ns-3 doxygen [45]. Písanie kódu začína vložením potrebných modulov - hlavičkových súborov s príponou `*.h`. Moduly sa zahŕňajú slovom `#include` a názvom daného hlavičkového súboru do časti „vkladanie modulov“. Vzhľadom k technológiám použitým v laboratórnej úlohe sú požité nasledujúce moduly:

```
#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/basic-energy-source.h"
#include "ns3/simple-device-energy-model.h"
#include <iostream>
```

Keď sú všetky moduly zahrnuté, tak sa musí definovať menný priestor (namespace). Menný priestor v NS-3 simulátore sa nazýva `ns3`. Implementáciou bude zaistené, že nemusí byť zadávaný ns3:: operátor pred celý kód NS-3 pred použitím. Použitý bude tiež menný priestor `std`. Vložený je nasledujúci kód:

```
using namespace ns3;
using namespace std;
```

Ďalej je definovaná funkcia, ktorá zaznamenáva informácie počas simulácie a v prípade chyby zužuje oblasť, kde danú chybu hľadať. Logovanie je vo východziom stave vypnuté a musí sa zapnúť. Pre aktiváciu logovania je použitý nasledujúci kód:

```
NS_LOG_COMPONENT_DEFINE ("WiFi");
```

Pred funkciou `int main(){};` z dôvodu animácií pomocou programu NetAnim a zobrazovania pozícii uzlov (je vysvetlené neskôr) je potrebné vložiť nasledujúci kód:

```
AnimationInterface * anim = 0;
```

Zaznamenávanie pohybu Wi-Fi staníc zabezpečuje nasledujúci kód. Druhá časť kódu, ktorý zaznamenáva trajektóriu uzlov je vložená a popísaná ku koncu laboratórnej úlohy.

```
void
CourseChange (std::string context, Ptr<const MobilityModel> model)
{
  Vector position = model->GetPosition ();
  NS_LOG_UNCOND (context <<
    " x = " << position.x << ", y = " << position.y);
}
```

Ako ďalší je pred hlavnú funkciu vložený kód pre meranie RTT (Round Trip Time).

```
static void PingRtt (std::string context, Time rtt)
{
  std::cout << context << " " << rtt << std::endl;
}
```

V predpripravenom súbore sa nachádza kód, ktorý slúži na demonštráciu odďaľovania staníc od prístupového bodu pri nastavení rôznych štandardov Wi-Fi.

V konzole bude zobrazené, na akú vzdialenosť sú schopné stanice signál prijímať v interiéri a exteriéri. Výpis rôznych štandardov sa nastavuje pomocou `boolean` hodnôt `true` alebo `false`. Vzdialenosť je nastavovaná volaním funkcie `NodeDistanceAccessPoint(double distance)`.

Skript je písaný v programovacom jazyku C++. Na jeho spustenie je potrebné použiť funkciu `int main(){}`, ktorá bude spustená ako prvá. Ďalej je definovaná premenná `verbose`, ktorá nadobúda hodnoty `true` alebo `false`. Na jej základe sú povolené alebo zakázané logy, záznamy, aplikácie `UdpEchoKlient` a `UdpEchoServer`. Sú to výpisy správ zobrazujúce prijímané a odosielané pakety. Ďalej je definovaný počet staníc CSMA (nCsma) a počet Wi-Fi staníc (nWifi). Kód sa vkladá pod metódu `NodeDistanceAccessPoint()`.

```
int main (int argc, char *argv[])
{
 bool verbose = true;
 uint32_t nCsma = 4;
 uint32_t nWifi = 4;

}
```

Následne je pod definíciou počtu Wi-Fi staníc vložená podmienka, ktorá obmedzuje počet možných vložených staníc na 18. Pri vyššom počte by bola simulácia výpočetne náročná a neprehľadná. Ďalší blok kódu aktivuje alebo deaktivuje logovanie.

```
 if (nWifi > 18)
  {
   std::cout << "Pocet uzlov " << nWifi <<
                  " presahuje povoleny pocet" << std::endl;
   exit (1);
  }
```

```
if (verbose)
  {
   LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
   LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
  }
```

Na vytvorenie dvoch uzlov prepojených technológiou point-to-point je vložený nasledujúci kód. Použitá je trieda `NodeContainer`.

```
NodeContainer p2pNodes;
p2pNodes.Create (2);
```

Následne sa nastavujú parametre linky point-to-point. Vytvorená je linka s rýchlosťou 5 Mbps a oneskorením 2 ms. Na vytvorenie takejto linky je použitý `PointToPointHelper`. Linka je nainštalovaná medzi dva uzly.

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);
```

Následne je vytvorený kontajner uzlov, ktorý obsahuje uzly na zbernici LAN siete. LAN sieť je následne pripojená k jednému z point-to-point uzlov vybraného z kontajneru uzlov CSMA/CD. Takto vytvorený uzol patrí do point-to-point spojenia a je súčasťou CSMA/CD LAN siete.

```
NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);
```

Podobne ako boli nastavené parametre point-to-point linky sú nastavované parametre LAN uzlov. Parametre sa nastavujú pomocou `CsmaHelper` volaním funkcie `SetChannelAttribute()`. Na koniec musia byť na danú linku nainštalované pomocou funkcie `Install()`. Popísané kroky zodpovedajú nasledujúcemu kódu:

```
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);
```

Následne sú vytvorené uzly, ktoré tvoria WiFi sieť. Uzol N0 bude tvoriť prístupový bod Wi-Fi siete (Access point). Prístupový bod bude pripojený do linky

point-to-point. Počet ďalších vytvorených uzlov závisí od nastavenia na začiatku kódu v položke `nWifi`. V tejto úlohe sú to štyri uzly N5, N6, N7 a N8.

```
NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi);
NodeContainer wifiApNode = p2pNodes.Get (0);
```

Nasledujúce riadky kódu umožnia prepojenie medzi bezdrôtovými uzlami a ich vzájomnú komunikáciu. Použitý je helper `YansWifiChannelHelper` a `YansWifiPhyHelper`. Z kódu vyplýva, že sa používa východzie nastavenie fyzickej vrstvy.

```
YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());
```

Po nastavení fyzickej vrstvy bude nastavená spojová vrstva. Použitý je helper, ktorý nezohľadňuje QoS. K tomuto účelu je použitý `NqosWifiMacHelper`. Metóda `SetStandard()` umožňuje nastaviť práve používaný štandard Wi-Fi s rôznymi podporovanými prenosovými rýchlosťami. Nastavovanie pomocou metódy `SetRemoteStationManager()`, umožňuje vybrať, aký algoritmus na prenos dát bude použitý.

```
WifiHelper wifi = WifiHelper::Default ();
wifi.SetStandard (WIFI_PHY_STANDARD_80211a);
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();
```

Ako ďalšie je nastavované SSID (identifikátor Wi-Fi siete) a MAC adresa. V tomto scenári je vypnuté aktívne skenovanie nastavením „`ActiveProbing`" na hodnotu `false`, čím sa aktivuje pasívne skenovanie. Pasívne skenovanie je pomalšie ako aktívne, uzol musí čakať na príchod beacon rámca. Beacon rámce sú posielané periodicky prístupovým bodom (Access Point). SSID je v tomto prípade „`VUTBR`". Vložením tohoto kódu je dokončená konfigurácia staníc, ktoré nebudú slúžiť ako prístupové body, ale ako stanice STA s použitím infraštruktúrnej BSS.

```
Ssid ssid = Ssid ("VUTBR");
mac.SetType ("ns3::StaWifiMac",
        "Ssid", SsidValue (ssid),
        "ActiveProbing", BooleanValue (false));
```

Po konfigurácii parametrov sa nastavenia nainštalujú na jednotlivé uzly.

```
NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);
```

Nasleduje konfigurácia prístupového bodu. Východzie parametre sú zmenené pomocou `WifiMacHelper`. Nasledujúci kód vytvorí MAC vrstvu, ktorá je špecifická pre prístupový bod.

```
mac.SetType ("ns3::ApWifiMac",
        "Ssid", SsidValue (ssid));
```

Aby prístupový bod zdieľal rovnaký typ fyzickej vrstvy a mohol komunikovať s vytvorenými stanicami, musia sa definované parametre nainštalovať.

```
NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, wifiApNode);
```

Prístupový bod musí byť stacionárny, stanice sa môžu aj nemusia pohybovať. V tomto scenári budú stanice mobilné. Využitý je `MobilityHelper`. Pomocou tohto helperu sú nastavené parametre jak prístupovému bodu tak jednotlivým staniciam. V tomto prípade sa používa 2D mriežka.

```
MobilityHelper mobility;

mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                        "MinX", DoubleValue (0.0),
                        "MinY", DoubleValue (0.0),
                        "DeltaX", DoubleValue (5.0),
                        "DeltaY", DoubleValue (10.0),
                        "GridWidth", UintegerValue (3),
                        "LayoutType", StringValue ("RowFirst"));
```

Aktuálne sú všetky uzly staticky umiestnené. Na ich pohyb je použitá funkcia `RandomWalkMobilityModel`. Touto funkciou bude docielené, že stanice sa budú pohybovať v určitom priestore definovanom súradnicami štvorca. Následne bude model nainštalovaný pre STA stanice.

```
mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
                           "Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)));
mobility.Install (wifiStaNodes);
```

Pre nastavenie statickosti prístupového bodu je použitý `MobilityHelper`. Pomocou neho je nastavená konštantná pozícia prístupového bodu. Znovu musí byť mobilita nainštalovaná na potrebný uzol.

```
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (wifiApNode);
```

V tomto momente je vytvorená topológia. Aby sa mohli priraďovať IP adresy a smerovať vo vytvorenej topológii musí sa nainštalovať internetový protokol. Na tento účel bude použitý `InternetStackHelper`.

```
InternetStackHelper stack;
stack.Install (csmaNodes);
stack.Install (wifiApNode);
stack.Install (wifiStaNodes);
```

Použité sú IPv4 adresy . Pre Wi-Fi sieť to je rozsah 192.168.10.128/26 pre point-to-point linku 192.168.10.0/26 a pre LAN je použitý rozsah 192.168.10.64/26. Na priradenie adries k rozhraniam je použitý helper `Ipv4AddressHelper`.

```
Ipv4AddressHelper address;

address.SetBase ("192.168.10.0", "255.255.255.192");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);

address.SetBase ("192.168.10.64", "255.255.255.192");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);

address.SetBase ("192.168.10.128", "255.255.255.192");
address.Assign (staDevices);
address.Assign (apDevices);
```

Prístupovému je nastavená batéria a pracovný prúd. K tomuto účelu je využitá trieda `BasicEnergySource` a `SimpleDeviceEnergyModel`. Nakoniec sa vytvorený model nainštaluje na konkrétny uzol N0 (prístupový bod).

```
Ptr<BasicEnergySource> energySource = CreateObject<BasicEnergySource>();
Ptr<SimpleDeviceEnergyModel> energyModel =
CreateObject<SimpleDeviceEnergyModel>();

energySource->SetInitialEnergy (700);
energyModel->SetEnergySource (energySource);
energySource->AppendDeviceEnergyModel (energyModel);
energyModel->SetCurrentA (20);

wifiApNode.Get (0)->AggregateObject (energySource);
```

Echo server je nastavený na uzol číslo N4 a N9 . Na základe kódu vidieť, že server sa bude nachádzať vždy na posledných dvoch pridaných uzloch Wi-Fi.

```
UdpEchoServerHelper echoServer (7);

ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));

ApplicationContainer serverApps1 = echoServer.Install (csmaNodes.Get (nCsma-1));
serverApps1.Start (Seconds (3.0));
serverApps1.Stop (Seconds (10.0));
```

Echo klient sa bude nachádzať na Wi-Fi staniciach (uzol N7 a N8). Klient sa bude nachádzať vždy na posledných dvoch pridaných uzloch LAN. Tiež je potrebné vytvoriť `ApplicationContainer`, z kadiaľ budú pakety posielané.

```
UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 7);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

UdpEchoClientHelper echoClient1 (csmaInterfaces.GetAddress (nCsma-1), 7);
echoClient1.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient1.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient1.SetAttribute ("PacketSize", UintegerValue (1024));
```

```
ApplicationContainer clientApps = echoClient.Install (wifiStaNodes.Get (nWifi - 1));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));

ApplicationContainer clientApps1 = echoClient.Install (wifiStaNodes.Get (nWifi - 2));
clientApps1.Start (Seconds (4.0));
clientApps1.Stop (Seconds (10.0));
```

Po vytvorení aplikácie posielajúcej ECHO protokol je vytvorená aplikácia, ktorá posiela ping. Najskôr je vložený kód na vytvorenie aplikácie na strane serveru, kde je použitý `OnOffHelper`. Na strane klienta je vytvorený kontajner pre príjem paketov.

```
InetSocketAddress ca = InetSocketAddress (csmaInterfaces.GetAddress (nCsma-2));
OnOffHelper onoff = OnOffHelper ("ns3::Ipv4RawSocketFactory", ca);
onoff.SetConstantRate (DataRate (15000));
onoff.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer apps = onoff.Install (csmaNodes.Get (nCsma-2));
apps.Start (Seconds (1.0));
apps.Stop (Seconds (10.0));
```

Ďalším krokom je vytvorenie aplikácie, kde sú pridávané jednotlivé stanice, ktoré posielajú ping na uzol N3 (192.168.10.67). Použitý je `V4PingHelper`. Ping je posielaný uzlami N2, N5 a N6 na uzol N3.

```
NS_LOG_INFO ("Create pinger");
V4PingHelper ping = V4PingHelper (csmaInterfaces.GetAddress (nCsma-2));
NodeContainer pingers;
pingers.Add (wifiStaNodes.Get (nWifi-3));
pingers.Add (wifiStaNodes.Get (nWifi-4));
pingers.Add (csmaNodes.Get (nCsma-3));
apps = ping.Install (pingers);
apps.Start (Seconds (2.0));
apps.Stop (Seconds (5.0));
```

Smerovanie sa v sieti aktivuje pomocou `Ipv4GlobalRoutingHelper`, ktorý je za smerovanie zodpovedný, vytvorí smerovacie tabuľky a potrebné náležitosti. `Ipv4GlobalRoutingHelper` má implementovaný protokol OSPF.

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Ďalším dôležitým krokom je zastavenie simulátoru v určitom čase, aby prístupový bod negeneroval beacon rámce donekonečna a simulácia by bola nekonečná. Nasledujúcim riadkom kódu je docielené, že simulácia skončí po 10 sekundách, čo bude následne vidieť v programe NetAnim.

```
Simulator::Stop (Seconds (10.0));
```

Aby bol tok paketov zaznamenávaný do *.pcap súborov, pre každý uzol, tak je použitá metóda `EnablePcap`, ktorá pre každé rozhranie uzlu vytvorí trasovací súbor a umiestni ho do vytvorenej zložky `/ns-allinone-3.21/ns-3.21/laboratorna_uloha`.

```
pointToPoint.EnablePcapAll ("laboratorna_uloha/WiFi");
phy.EnablePcap ("laboratorna_uloha/WiFi", apDevices.Get (0));
csma.EnablePcap ("laboratorna_uloha/WiFi", csmaDevices.Get (0), true);
```

Nasledujúci kód vytvorí súbor `Wifi.routes`, kde sú záznamy o smerovaní-smerovacie tabuľky. Súbor sa tak isto nachádza v zložke `ns-allinone3.21/ns-3.21/laboratorna_uloha`. Smerovacie tabuľky sú zaznamenané v čase 5 s. V tomto prípade nezávisí od času, smerovacie tabuľky budú rovnaké po celú dobu simulácie. Na ich generovanie je použitý `OutputStreamWrapper`.

```
Ipv4GlobalRoutingHelper str;
Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper>
("laboratorna_uloha/WiFi.routes", std::ios::out);
str.PrintRoutingTableAllAt (Seconds (5), routingStream);
```

Na vypísanie súradníc niektorého z uzlov je vložený nasledujúci kód, ktorý je pokračovaním bloku kódu vloženého pred hlavnou funkciou `int main(){};`.

```
std::ostringstream oss;
oss <<
  "/NodeList/" << wifiStaNodes.Get (nWifi -3)->GetId () <<
  "/$ns3::MobilityModel/CourseChange";

Config::Connect (oss.str (), MakeCallback (&CourseChange));
```

Na vizualizáciu výslednej topológie s tokom paketov je použitý nástroj NetAnim. Pred hlavnou funkciou `int main(){}` je už vložený riadok kódu

súvisiaci s animáciou. Najskôr je vytvorený objekt `anim` z `AnimationInterface` a následne je určená pozícia uzlov a priradený popis, farba a súradnice každému z nich.

Pre zobrazenie informácií o posielaných rámcoch, správach a paketoch sú povolené `PacketMetadata` nastavením hodnoty na `true`.

```
anim = new AnimationInterface("laboratorna_uloha/animace_WiFi.xml");

anim->UpdateNodeDescription(wifiStaNodes.Get(0), "N5");
anim->SetConstantPosition(wifiStaNodes.Get(0), 6, 10);

anim->UpdateNodeDescription(wifiStaNodes.Get(1), "N6");
anim->SetConstantPosition(wifiStaNodes.Get(1), 9, 10);

anim->UpdateNodeDescription(wifiStaNodes.Get(2), "N7");
anim->SetConstantPosition(wifiStaNodes.Get(2), 12, 10);

anim->UpdateNodeDescription(wifiStaNodes.Get(3), "N8");
anim->SetConstantPosition(wifiStaNodes.Get(3), 15, 10);

anim->UpdateNodeDescription(p2pNodes.Get(0), "Access point");
anim->UpdateNodeColor(p2pNodes.Get(0), 0, 0, 255);

anim->UpdateNodeDescription(csmaNodes.Get(0), "N1_P2P");
anim->SetConstantPosition (csmaNodes.Get(0), 15, 10);
anim->UpdateNodeColor(p2pNodes.Get(0), 0, 0, 255);

anim->UpdateNodeDescription(csmaNodes.Get(1), "N2");
anim->SetConstantPosition (csmaNodes.Get(1), 22.5, 14);

anim->UpdateNodeDescription(csmaNodes.Get(2), "N3");
anim->SetConstantPosition (csmaNodes.Get(2), 22.5, 9);

anim->UpdateNodeDescription(csmaNodes.Get(3),"N4");
anim->SetConstantPosition (csmaNodes.Get(3), 22.5, 4);

anim->UpdateNodeDescription(csmaNodes.Get(4),"N9");
anim->SetConstantPosition (csmaNodes.Get(4), 22.5, -1);

anim->EnablePacketMetadata (true);
```

Ďalším krokom je vloženie sondy, ktorá bude merať čas RTT. Kód je pokračovaním metódy `PingRTT()`.

```
Config::Connect ("/NodeList/*/ApplicationList/*/$ns3::V4Ping/Rtt",MakeCallback
(&PingRtt));
```

Pred funkciu `Run()`, ktorá spúšťa simulátor je vložený modul `FlowMonitor`. Tento modul sleduje tok paketov na zvolených uzloch. Základnými sledovanými parametrami sú počet odoslaných a prijatých paketov, meškanie (delay), jitter a priepustnosť. Modul sa aplikuje pomocou `FlowMonitorHelperu`.

```
FlowMonitorHelper flowHelper;
Ptr<FlowMonitor> flowMonitor;
flowMonitor = flowHelper.InstallAll(); Simulator::Stop (Seconds (11));
Simulator::Run ();
```

Na spustenie simulátoru je použitá funkcia `Simulator::Run()`, ktorá zistí, aké udalosti sú naplánované a vykoná ich. Po funkcii `Run()` je vložená druhá časť kódu pre záznam parametrov do konzole a vytvorenie súboru s príponou `*.xml` v zložke `laboratorna_uloha`. Ukončenie simulátora prebieha pomocou funkcie `Simulator::Destroy()`. Volaná je funkcia `NodeDistanceAccessPoint()`, ktorou sa nastavuje vzdialenosť staníc v metroch od prístupového bodu a následný percentuálny dosah staníc v interiéri a exteriéri pre rôzne Wi-Fi štandardy. Všetky stanice budú posunuté do rovnakej vzdialenosti od prístupového bodu.

```
flowMonitor->CheckForLostPackets ();
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>
(flowHelper.GetClassifier ());
std::map<FlowId, FlowMonitor::FlowStats> stats = flowMonitor->GetFlowStats ();
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i =
                    stats.begin (); i != stats.end (); ++i)
  {

Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
NS_LOG_UNCOND( "\n Flow ID: " << i->first << " Src Addr: " << t.sourceAddress << "
Dst Addr: " << t.destinationAddress);
NS_LOG_UNCOND( " Tx Bytes: " << i->second.txBytes);
NS_LOG_UNCOND( " Rx Packets: " << i->second.rxPackets);
NS_LOG_UNCOND( " Mean Delay: " << i->second.delaySum.GetSeconds () / (i-
>second.rxPackets) * 1000 << " ms");
```

```
NS_LOG_UNCOND( " Mean Jitter: " << i->second.jitterSum.GetSeconds () / (i-
>second.rxPackets - 1) * 1000 << " ms");
NS_LOG_UNCOND( " Throughput: " << i->second.rxBytes * 8.0 / (i-
>second.timeLastRxPacket.GetSeconds () - i->second.timeFirstTxPacket.GetSeconds
()) / 1024 << " Kbps");
  }
flowMonitor->SerializeToXmlFile("laboratorna_uloha/uloha1Flowmon", true, true);

Simulator::Destroy ();
NodeDistanceAccessPoint(9);

return 0;
}
```

Po týchto krokoch je hotový skript a simulácia môže byť spustená. Po spustení sa zobrazí výpis v konzole, ktorý zobrazuje, ako sa pohybovali jednotlivé stanice a taktiež v akých časoch poslali klienti serverom paket a kedy boli prijaté pre oba smery. Vytvorené sú štyri súbory `*.pcap`, ktoré je možné otvoriť v sieťovom analyzátore WireShark, súbor `animace_WiFi.xml`, ktorý zobrazuje graficky priebeh simulácie. Výstupom je aj súbor `WiFi.routes`, v ktorom sa nachádzajú smerovacie tabuľky. V konzole sa objavia informácie o počte prijatých, odoslaných paketoch, meškaní, jitteri a priepustnosti v kb/s. Tiež sa zobrazí výpis o časoch RTT a dosah Wi-Fi staníc vzhľadom k prístupovému bodu v percentách.

6.1.1 Spustenie a zobrazenie výsledkov simulácie

Simulácia sa spúšťa tlačidlom Run rovnako ako výpis testovacieho reťazca do konzoly na začiatku simulácie. Simulácia sa následne spustí volaním funkcie `Simulator::Run()` volaním metód `Start()` a `Stop()` pri klientovi a serveri. Po realizácii všetkých udalostí sa simulácia prepne do stavu nečinnosti a zastaví sa aplikácia pre server aj klienta. Nakoniec prebehne vymazanie všetkých objektov simulácie pomocou funkcie `Simulator::Destroy()`.

V konzole sa zobrazí informácia o prenesených paketoch medzi klientom a serverom s časmi, v ktorých bol paket odoslaný a prijatý. Obr. 6.1.

```
At time 2s client sent 1024 bytes to 192.168.10.69 port 7
At time 2.02293s server received 1024 bytes from 192.168.10.132 port 49153
At time 2.02293s server sent 1024 bytes to 192.168.10.132 port 49153
At time 2.0376s client received 1024 bytes from 192.168.10.69 port 7
At time 4s client sent 1024 bytes to 192.168.10.68 port 7
At time 4.01796s server received 1024 bytes from 192.168.10.131 port 49153
At time 4.01796s server sent 1024 bytes to 192.168.10.131 port 49153
At time 4.03362s client received 1024 bytes from 192.168.10.68 port 7
```

Obr. 6.1 Výpis poslaných a prijatých bajtov z konzole

Ďalším výstupom sú informácie o dátovom toku medzi jednotlivými uzlami. Prvý tok s ID: 1 zobrazuje tok dát medzi adresami 192.168.10.132/26 a 192.168.10.69/26 (uzol N8 a N9), tok dát s ID: 2 zobrazuje opačný tok dát, medzi adresami 192.168.10.68/26 a 192.168.10.131/26. Druhý tok s ID: 3 a ID: 4 zobrazuje prenos medzi adresami 192.168.10.131/26 a 192.168.10.68/26 (uzol N7 a N4) oboma smermi. Obr. 6.2.

```
Flow ID: 1
Tx Bytes: 1052
Rx Bytes: 1052
Tx Packets: 1
Rx Packets: 1
Mean Delay: 22.928 ms
Mean Jitter: -nan ms
Throughput: 358.46 Kbps
```

```
Flow ID: 2
Tx Bytes: 1052
Rx Bytes: 1052
Tx Packets: 1
Rx Packets: 1
Mean Delay: 14.668 ms
Mean Jitter: -nan ms
Throughput: 560.32 Kbps
```

```
Flow ID: 3
Tx Bytes: 1052
Rx Bytes: 1052
Tx Packets: 1
Rx Packets: 1
Mean Delay: 17.955 ms
Mean Jitter: -nan ms
Throughput: 457.741 Kbps
```

```
Flow ID: 4
Tx Bytes: 1052
Rx Bytes: 1052
Tx Packets: 1
Rx Packets: 1
Mean Delay: 15.668 ms
Mean Jitter: -nan ms
Throughput: 524.556 Kbps
```

Obr. 6.2 Jednotlivé toky dát

V adresári `/ns-allinone-3.21/ns-3.21/laboratorna_uloha`, je vytvorený súbor `WiFi.routes`, ktorý zobrazuje smerovaciu tabuľku, ktorá obsahuje zoznam adries a k ním prislúchajúcu východziu bránu a rozhranie, na ktoré majú byť pakety s danou cieľovou adresou posielané. Na obrázku Obr. 6.3 je zobrazená len časť z tohto súboru.

WiFi.routes ×

```
Node: 0 Time: 5s Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Ipv4StaticRouting
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
127.0.0.0       0.0.0.0         255.0.0.0       U     0      -      -   0
192.168.10.0    0.0.0.0         255.255.255.192 U     0      -      -   1
192.168.10.128  0.0.0.0         255.255.255.192 U     0      -      -   2
  Priority: -10 Protocol: ns3::Ipv4GlobalRouting
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.10.2    192.168.10.2    255.255.255.255 UH    -      -      -   1
192.168.10.128  0.0.0.0         255.255.255.192 U     -      -      -   2
192.168.10.64   192.168.10.2    255.255.255.192 UG    -      -      -   1
127.0.0.0       192.168.10.129  255.0.0.0       UG    -      -      -   2
127.0.0.0       192.168.10.130  255.0.0.0       UG    -      -      -   2
127.0.0.0       192.168.10.131  255.0.0.0       UG    -      -      -   2
127.0.0.0       192.168.10.132  255.0.0.0       UG    -      -      -   2
```

Obr. 6.3 Smerovacia tabuľka

Taktiež sa vytvorili štyri trasovacie súbory pre obidva uzly nachádzajúce sa v spojení point-to-point. Cez tieto dva uzly ide celý tok paketov, takže nie je potrebné vypisovať tok z ostatných uzlov. Sú to súbory `WiFi-0-0.pcap`, `WiFi-0-1.pcap`, `WiFi-1-0.pcap` a `WiFi-1-1.pcap`. V súbore `WiFi-0-0.pcap`, Obr. 6.4 sa nachádza nasledujúci výpis. Použitý protokol je ECHO a ICMP.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.130	192.168.10.67	ICMP	86	Echo (ping) request
2	0.001955	192.168.10.129	192.168.10.67	ICMP	86	Echo (ping) request
3	0.004355	192.168.10.67	192.168.10.130	ICMP	86	Echo (ping) reply
4	0.006247	192.168.10.132	192.168.10.69	ECHO	1054	Request
5	0.006260	192.168.10.67	192.168.10.129	ICMP	86	Echo (ping) reply
6	0.028854	192.168.10.69	192.168.10.132	ECHO	1054	Response

Obr. 6.4 Obsah súboru WiFi-0-0.pcap

Na obrázku Obr. 6.5 je zobrazená časť zo súboru `WiFi-0-1.pcap`, v ktorom je vidieť periodické vysielanie beacon rámca prístupovým bodom, použitie protokolu ARP (Address Resolution Protocol) na zistenie MAC adresy podľa IP adresy a následnú odpoveď. Zobrazené je SSID a typ protokolu posielajúci beacon rámce (802.11).

No.	Time	Source	Destination	Protocol	Length	Info
34	1.638375	00:00:00_00:00:0c	Broadcast	802.11	57	Beacon frame, SN=20, FN=0, Flags=o......., BI=100, SSID=VUTBR
35	1.740775	00:00:00_00:00:0c	Broadcast	802.11	57	Beacon frame, SN=21, FN=0, Flags=o......., BI=100, SSID=VUTBR
36	1.843175	00:00:00_00:00:0c	Broadcast	802.11	57	Beacon frame, SN=22, FN=0, Flags=o......., BI=100, SSID=VUTBR
37	1.945575	00:00:00_00:00:0c	Broadcast	802.11	57	Beacon frame, SN=23, FN=0, Flags=o......., BI=100, SSID=VUTBR
38	2.004087	00:00:00_00:00:09	Broadcast	ARP	64	Who has 192.168.10.133? Tell 192.168.10.130
39	2.004103		00:00:00_00:00:09 (RA)	802.11	14	Acknowledgement, Flags=o.......
40	2.004181	00:00:00_00:00:09	Broadcast	ARP	64	Who has 192.168.10.133? Tell 192.168.10.130
41	2.004462	00:00:00_00:00:0c	00:00:00_00:00:09	ARP	64	192.168.10.133 is at 00:00:00:00:00:0c
42	2.004634		00:00:00_00:00:0c (RA)	802.11	14	Acknowledgement, Flags=o.......
43	2.004852	192.168.10.130	192.168.10.67	ICMP	120	Echo (ping) request id=0x0000, seq=0/0, ttl=64
44	2.004868		00:00:00_00:00:09 (RA)	802.11	14	Acknowledgement, Flags=o.......

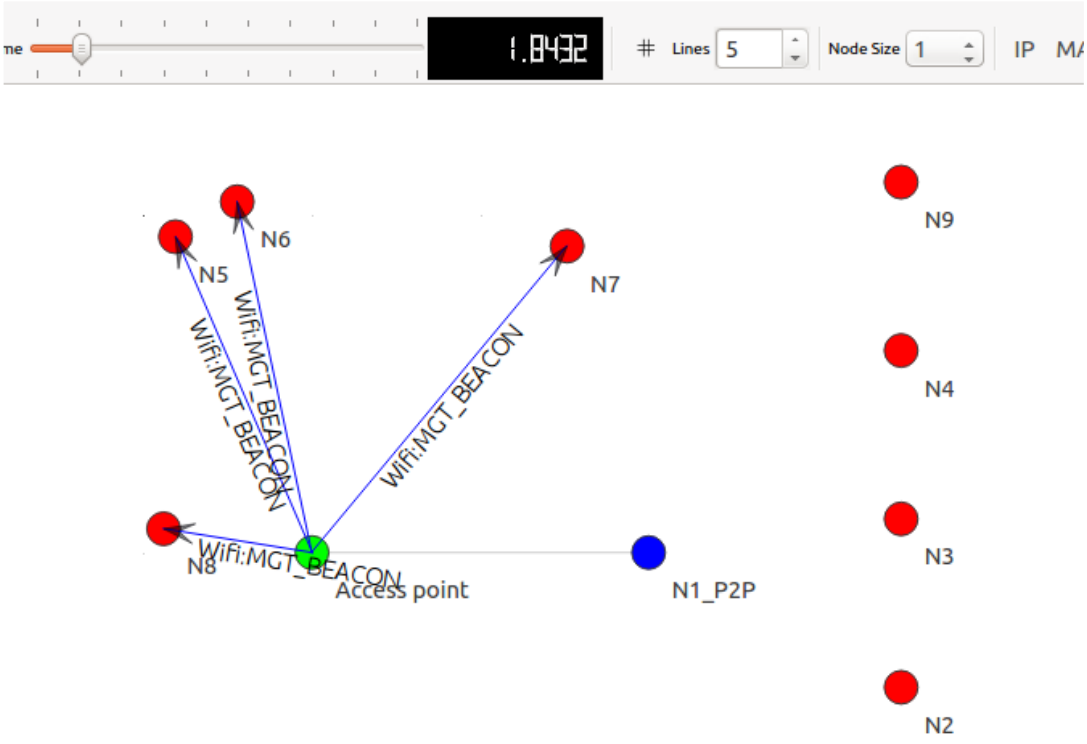
Obr. 6.5 Obsah súboru WiFi-0-1.pcap

Po otvorení súboru `animace_OSPF.xml` v programe NetAnim sa zobrazia najskôr Wi-Fi stanice s prístupovým bodom a uzlami v spojení point-to-point (N5, N6, N7, N8, Access point, N1_P2P). V čase 0,25 s sa zobrazia stanice lokálnej siete LAN po asociácii Wi-Fi staníc s prístupovým bodom. LAN stanice nemajú v programe NetAnim vykreslené linky medzi sebou. Linky medzi nimi existujú avšak NetAnim ich nedokáže pri LAN a technológií CSMA/CD vykresliť.

Po skončení simulácie a preskúmaní toku paketov je vhodné zobraziť trajektóriu uzlov, ktorá je vypísaná v konzole programu Eclipse. Akcia sa vykonáva kliknutím na `Show Properties Tree`, nastavením `Show Node Trajectory` na `true` pri uzloch Node 6, 7, 8, 9.

Pred spustením simulácie je vhodné vypnúť mriežku, zobrazujúcu súradnice a zväčšiť topológiu lupou kvôli prehľadnejšiemu zobrazovaniu správ, rámcov a paketov. V simulácii je vidieť všetky toky zobrazené v trasovacích súboroch. Prístupový bod je zobrazený zelenou farbou a uzol k nemu pripojený linkou point-to-point modrou farbou. Výsledná topológia z programu NetAnim je zobrazená na obrázku Obr. 6.6.

V konzole sa zobrazí čas RTT pre uzly N2, N6 a N7. Doba RTT medzi dvoma uzlami v sieti LAN je niekoľkonásobne menšia ako RTT medzi LAN a Wi-Fi sieťou.



Obr. 6.6 Výsledná topológia v programe NetAnim

6.2 Tvorba simulačného modelu pre laboratórnu úlohu Smerovací protokol OSPF

Topológiu zobrazenú na obrázku Obr. 5.5 je potrebné vpísať do skriptu v jazyku C++, ktorý bude spustitelný v simulačnom nástroji NS-3. Laboratórna úloha je vytvorená pomocou dokumentácie NS-3 [41] a NS-3 doxygen [45]. Písanie kódu začína vložením potrebných modulov - hlavičkových súborov s príponou `*.h` knižnice NS-3. Moduly sa zahŕňajú slovom `#include` a názvom daného hlavičkového súboru. Vzhľadom k technológiám použitým v laboratórnej úlohe sú používané nasledujúce moduly:

```
#include <iostream>
#include <fstream>
#include <string>
#include <cassert>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/gnuplot.h"
#include "ns3/stats-module.h"
```

Po zahrnutí modulov musí byť deklarovaný menný priestor (namespace). Menný priestor v NS-3 simulátore sa nazýva `ns3`. Po jeho implementácii je zaistené, že nie je potrebné zadávať `ns3::` operátor pred celý kód NS-3 pred použitím. Tiež je použitý menný priestor `std`.

```
using namespace ns3;
using namespace std;
```

Ďalej je definovaná funkcia, ktorá zaznamenáva informácie počas simulácie a v prípade chyby zužuje oblasť, kde danú chybu hľadať. Logovanie je vo východziom stave vypnuté a musí sa zapnúť. Logovanie sa aktivuje nasledujúcim kódom:

```
NS_LOG_COMPONENT_DEFINE ("OSPF");
```

Pred funkciou `int main(){}` je z dôvodu animácií pomocou programu NetAnim, zobrazovania pozícii uzlov (je vysvetlené neskôr) a pre potreby tvorby grafu pomocou `Gnuplot` je vložený nasledujúci kód:

```
AnimationInterface * anim = 0;
Gnuplot2dDataset txAllDataset;
```

Pred hlavnou funkciou sú definované premenné a metódy, pomocou ktorých sú vykresľované grafy. Sú to metódy `sent()` a `save()`.

```
int size = 0;
double startTime = 2.0;
double endTime = 10.0;
double section = 0.5;

void sent(string path, Ptr<Packet const> packet, Ptr<Ipv4> ipv4,
unsigned int p) {
  if (packet->GetSize() >= 100) {
     size = size + packet->GetSize();
  }
}

void save(double time) {
  dataset.Add(time, (double)size/section*8/(1024*1024));
  size = 0;
}
```

Keďže je skript napísaný v programovacom jazyku C++. Na jeho spustenie je použitá funkcia `int main(){}`, ktorá bude spustená ako prvá. Do tela tejto funkcie je postupne vkladaný celý zdrojový kód s topológiou siete, parametrami liniek a konfiguráciou. Na definovanie sieťových prvkov, IP adries, protokolov sú použité helpery, ktoré uľahčujú prácu. Najskôr sú vložené funkcie na zaznamenávanie činnosti simulátoru `LogComponentEnable`.

```
int main (int argc, char *argv[])
{
   #if 0
   LogComponentEnable ("GlobalRoutingHelper", LOG_LOGIC);
   LogComponentEnable ("GlobalRouter", LOG_LOGIC);
   #endif
}
```

Následne je vložená funkcia `Ipv4GlobalRouting::RespondTo InterfaceEvent`, ktorá má hodnotu nastavenú na `true`. Táto funkcia je kritická na fungovanie laboratórnej úlohy. Ak by sa táto časť kódu zakomentovala, protokol OSPF by nebol schopný reagovať na výpadky liniek a iné zmeny v topológii počas simulácie. Po výpadku niektorej z liniek alebo zmene metriky by boli pakety zahadzované.

```
Config::SetDefault ("ns3::Ipv4GlobalRouting::RespondToInterfaceEvents",
BooleanValue (true));
```

Zavedená je premenná `verbose`, ktorá nadobúda hodnoty `true` alebo `false`. Premenná `verbose` je použitá na rozhodovanie podmienky, či bude použitý UDP protokol so službou TFTP alebo TCP protokol so službou HTTPS.

```
bool verbose = (true);
```

Počas celého vkladania častí kódu sú zakomponované komentáre, ktoré sa zobrazia v logovacom súbore, aby bolo možné rýchlejšie vyhľadať prípadnú chybu pri simulácii. Ďalej tieto časti kódu nebudú popisované.

```
NS_LOG_UNCOND ("Projekt OSPF");
NS_LOG_INFO ("Vytvorenie uzlu.");
```

Následne je vytváraná topológia siete podľa obrázku Obr. 5.5. Najskôr je vytvorený potrebný počet uzlov pomocou helperu `NodeContainer` pre spojenie point-to-point. Vytvorených je potrebných 8 uzlov. Ak sa vytvorí väčší počet uzlov, simulácia bude fungovať, avšak smerovacia tabuľka obsahuje redundantné informácie o neexistujúcich uzloch.

```
NodeContainer p2pNodes;
p2pNodes.Create (8);
```

Po vytvorení jednotlivých uzlov, musia byť medzi sebou previazané a tiež musia byť vytvorené jednotlivé linky. K tomuto účelu je použitý `NodeContainer`. Pre každú linku je vytvorený vlastný kontajner. Linka medzi uzlom N0 a N1 je v skripte označená ako n0n1 analogicky to platí pre všetky linky. Takýmto

spôsobom je vytvorených 11 kontajnerov medzi jednotlivými uzlami. Po vykonaní kódu sú vytvorené kontajnery dvojíc uzlov pre topológiu zobrazenú na obrázku Obr. 5.5. Uzly sa budú volať pomocou metódy `Get()` a ako argument je použité číslo uzlu.

```
NodeContainer n0n1 = NodeContainer (p2pNodes.Get (0), p2pNodes.Get (1));
NodeContainer n1n7 = NodeContainer (p2pNodes.Get (1), p2pNodes.Get (7));
NodeContainer n1n2 = NodeContainer (p2pNodes.Get (1), p2pNodes.Get (2));
NodeContainer n1n3 = NodeContainer (p2pNodes.Get (1), p2pNodes.Get (3));
NodeContainer n1n6 = NodeContainer (p2pNodes.Get (1), p2pNodes.Get (6));
NodeContainer n2n4 = NodeContainer (p2pNodes.Get (2), p2pNodes.Get (4));
NodeContainer n2n6 = NodeContainer (p2pNodes.Get (2), p2pNodes.Get (6));
NodeContainer n3n4 = NodeContainer (p2pNodes.Get (3), p2pNodes.Get (4));
NodeContainer n3n7 = NodeContainer (p2pNodes.Get (3), p2pNodes.Get (7));
NodeContainer n4n5 = NodeContainer (p2pNodes.Get (4), p2pNodes.Get (5));
NodeContainer n4n7 = NodeContainer (p2pNodes.Get (4), p2pNodes.Get (7));
```

Ďalej sú definované parametre jednotlivých liniek. Použitý je `PointToPointHelper`. Všetkým linkám je definovaná rýchlosť prenosu `DataRate` s hodnotou `512 kb/s` a oneskorením `Delay` na linke `0 ms`.

```
NS_LOG_INFO ("Vytvorenie kanalov.");
PointToPointHelper p2p;
p2p.SetDeviceAttribute ("DataRate", StringValue ("512kbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("0ms"));
```

Po nastavení parametrov linky je táto hodnota nainštalovaná medzi dvojice uzlov. Pre tento účel je použitý `NetDeviceContainer` a funkcia `Install()`. Ako parametre sú zadávané jednotlivé linky medzi zariadeniami z `NodeContaineru`.

```
NetDeviceContainer d0d1 = p2p.Install (n0n1);
NetDeviceContainer d1d2 = p2p.Install (n1n2);
NetDeviceContainer d1d3 = p2p.Install (n1n3);
NetDeviceContainer d1d6 = p2p.Install (n1n6);
NetDeviceContainer d1d7 = p2p.Install (n1n7);
NetDeviceContainer d2d4 = p2p.Install (n2n4);
NetDeviceContainer d2d6 = p2p.Install (n2n6);
NetDeviceContainer d3d4 = p2p.Install (n3n4);
NetDeviceContainer d3d7 = p2p.Install (n3n7);
NetDeviceContainer d4d5 = p2p.Install (n4n5);
NetDeviceContainer d4d7 = p2p.Install (n4n7);
```

Ďalším krokom po vytvorení topológie siete je nastavenie sady protokolov, ktorá je využívaná v laboratórnej úlohe. Použitá je sada protokolov TCP/IP. Všetky uzly sú vytvorené v kontajneri `p2pNodes`. Protokolová sada musí byť aplikovaná a rozšírená do celej topológie. Túto funkciu poskytuje `InternetStackHelper`. Protokolová sada je po vykonaní kódu nainštalovaná na všetky uzly v topológií (všetky fungujú ako smerovače OSPF).

```
InternetStackHelper internet;
internet.Install (p2pNodes);
```

Adresácia je riešená pomocou helperu `Ipv4AddressHelper`. Z toho vyplýva, že použité adresy sú typu Ipv4. Pre každú linku vytvorenú pomocou `NetDeviceContainer` je priradená Ipv4 adresa. Adresy sú zvolené tak, aby každá linka spadala do iného adresného rozsahu. Jednotlivým rozhraniam sú priradené Ipv4 adresy pomocou topology helperu automaticky po začatí simulácie. Najskôr je pre každú linku vytvorený kontajner s priradenou IP adresou pomocou `Ipv4InterfaceContainer`. Kód k popísaným krokom je nasledujúci:

```
NS_LOG_INFO ("Priradenie adries");
Ipv4AddressHelper ipv4;
ipv4.SetBase ("192.168.10.16", "255.255.255.240");
Ipv4InterfaceContainer i0i1 = ipv4.Assign (d0d1);
ipv4.SetBase ("192.168.10.112", "255.255.255.240");
Ipv4InterfaceContainer i1i2 = ipv4.Assign (d1d2);
ipv4.SetBase ("192.168.10.0", "255.255.255.240");
Ipv4InterfaceContainer i1i3 = ipv4.Assign (d1d3);
ipv4.SetBase ("192.168.10.160", "255.255.255.240");
Ipv4InterfaceContainer i1i7 = ipv4.Assign (d1d7);
ipv4.SetBase ("192.168.10.32", "255.255.255.240");
Ipv4InterfaceContainer i1i6 = ipv4.Assign (d1d6);
ipv4.SetBase ("192.168.10.80", "255.255.255.240");
Ipv4InterfaceContainer i2i4 = ipv4.Assign (d2d4);
ipv4.SetBase ("192.168.10.48", "255.255.255.240");
Ipv4InterfaceContainer i2i6 = ipv4.Assign (d2d6);
ipv4.SetBase ("192.168.10.64", "255.255.255.240");
Ipv4InterfaceContainer i3i4 = ipv4.Assign (d3d4);
ipv4.SetBase ("192.168.10.128", "255.255.255.240");
Ipv4InterfaceContainer i3i7 = ipv4.Assign (d3d7);
ipv4.SetBase ("192.168.10.96", "255.255.255.240");
Ipv4InterfaceContainer i4i5 = ipv4.Assign (d4d5);
ipv4.SetBase ("192.168.10.144", "255.255.255.240");
Ipv4InterfaceContainer i4i7 = ipv4.Assign (d4d7);
```

Ďalším krokom je nastavovanie metriky. OSPF protokol udržuje v smerovacích tabuľkách informácie o metrike ciest a následne podľa toho, ktorá cesta má najmenšiu metriku k cieľovej destinácii zvolí najvýhodnejšiu trasu na prenos paketu. Do logovacieho súboru sa zapíše, že nasledujúcim kódom je nastavovaná metrika. Metrika je priraďovaná pomocou funkcie `SetMetric(rozhranie, metrika)`. Metrika sa dá zvoliť rôzna pre každý smer prenosu. Východzia metrika je 20 pre všetky smery a rozhrania. Počas laboratórnej úlohy je metrika upravovaná tak, aby boli pakety smerované zadanou trasou pri riešení samostatných úloh.

```
NS_LOG_INFO ("Nastavenie metriky");

i0i1.SetMetric (0, 20);
i0i1.SetMetric (1, 20);
i1i2.SetMetric (0, 20);
i1i2.SetMetric (1, 20);
i1i3.SetMetric (0, 20);
i1i3.SetMetric (1, 20);
i1i6.SetMetric (0, 20);
i1i6.SetMetric (1, 20);
i1i7.SetMetric (0, 20);
i1i7.SetMetric (1, 20);
i2i4.SetMetric (0, 20);
i2i4.SetMetric (1, 20);
i2i6.SetMetric (0, 20);
i2i6.SetMetric (1, 20);
i3i4.SetMetric (0, 20);
i3i4.SetMetric (1, 20);
i3i7.SetMetric (0, 20);
i3i7.SetMetric (1, 20);
i4i5.SetMetric (0, 20);
i4i5.SetMetric (1, 20);
i4i7.SetMetric (0, 20);
i4i7.SetMetric (1, 20);
```

Aby bolo možné aktivovať smerovanie medzi rôznymi sieťami a bol umožnený prechod paketov sieťou, je použitý `Ipv4GlobalRoutingHelper`. Daný helper je použitý s výhodou pre danú laboratórnu úlohu, pretože má implementovaný protokol OSPF, na základe ktorého smeruje pakety v sieti.

Po jeho aktivácii má celá sieť funkcionality protokolu OSPF. Do logovacieho súboru je zapísané, že sa aktivovalo smerovanie. Kód na aktiváciu je nasledovný:

```
NS_LOG_INFO ("Aktivacia smerovania");

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Ďalší krok je vytvorenie aplikácie, ktorá bude pakety generovať a tiež aplikácie, ktorá bude pakety prijímať. Pre generovanie paketov je zvolená funkcia `OnOffHelper`. Aplikácia na generovanie paketov je nastavená na uzol číslo 5 s IP adresou 192.168.10.98. Veľkosť paketu je 1024 B, cieľová destinácia je nastavená na rozhraní i0i1, uzol N0 s IP adresou 192.168.10.17.

Port je na začiatku nastavený na náhodné číslo z rozsahu väčšieho ako 49151. Použitý je transportný protokol UDP. V laboratórnej úlohe je následne číslo portu zmenené na 69. Port 69 je rezervovaný pre protokol TFTP a nespoľahlivý protokol UDP. Preto v prvom prípade bude použitý `ns3::UdpSocketFactory`.

Zmena portu sa realizuje na zmenou hodnoty v premennej port: `uint16_t port = 52379`. Následne je zavedená premenná `verbose`, ktorá nadobúda hodnoty `true` alebo `false`. Na základe premennej `verbose` sa podľa podmienky nastavuje prenos UDP na porte 69 (TFTP) alebo prenos TCP na porte 443 (HTTPS). Služba HTTPS vyžaduje potvrdzovanie prijatých paketov a dochádza k segmentácii.

Pre príjem paketov je zvolený `PacketSinkHelper`. `PacketSinkHelper` a `OnOffHelper` majú nastavený rovnaký port. Ak sú aplikáciou posielané UDP pakety, tak sú tiež UDP pakety očakávané (použitie `ns3::UdpSocketFactory`). Platí to aj pre `ns3::TcpSocketFactory`. Pri vytváraní aplikácie pre generovanie a príjem paketov sa pracuje so soketom, čo je kombinácia IP adresy a portu. Premenná `verbose` je nastavovaná na začiatku skriptu. Ak je hodnota nastavená na `true`, aktivovaný je UDP protokol. Ak je premenná `verbose` nastavená na `false`, aktivovaný je TCP protokol. Aplikácia pre generátor aj príjemcu paketov používajúce protokol UDP je nasledovná:

```
if(verbose == true){

            NS_LOG_INFO ("Vytvorenie klientskej aplikacie");

            uint16_t port = 69;

          OnOffHelper onoffUdp ("ns3::UdpSocketFactory", InetSocketAddress
(i0i1.GetAddress (0), port));
          onoffUdp.SetAttribute ("PacketSize", UintegerValue (1024));
          onoffUdp.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
          onoffUdp.SetAttribute("OffTime",
StringValue("ns3::ConstantRandomVariable[Constant=0]"));
          ApplicationContainer apps = onoffUdp.Install (p2pNodes.Get (5));
          apps.Start (Seconds (1.0));
          apps.Stop (Seconds (10.0));

          NS_LOG_INFO ("Vytvorenie aplikacie na strane serveru");

          PacketSinkHelper sinkUdp ("ns3::UdpSocketFactory", Address
(InetSocketAddress (Ipv4Address::GetAny (), port)));
          apps = sinkUdp.Install (p2pNodes.Get (0));
          apps.Start (Seconds (1.0));
          apps.Stop (Seconds (10.0));
 }
```

Aplikácia pre generátor paketov používajúca TCP protokol je nasledovná:

```
else{
          NS_LOG_INFO ("Vytvorenie klientskej aplikacie");

          uint16_t port = 443;
          OnOffHelper onoffTcp ("ns3::TcpSocketFactory", InetSocketAddress
(i0i1.GetAddress (0), port));
          onoffTcp.SetAttribute ("PacketSize", UintegerValue (1024));
          onoffTcp.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
          onoffTcp.SetAttribute("OffTime",
StringValue("ns3::ConstantRandomVariable[Constant=0]"));
          ApplicationContainer apps = onoffTcp.Install (p2pNodes.Get (5));
          apps.Start (Seconds (1.0));
          apps.Stop (Seconds (10.0));
```

```
NS_LOG_INFO ("Vytvorenie aplikacie na strane serveru");

                PacketSinkHelper sinkTcp ("ns3::TcpSocketFactory", Address
(InetSocketAddress (Ipv4Address::GetAny (), port)));
                apps = sinkTcp.Install (p2pNodes.Get (0));
                apps.Start (Seconds (1.0));
                apps.Stop (Seconds (10.0));
  }
```

Následne sú nastavované výpadky liniek. Linka medzi uzlami N3 a N4 vypadne v čase 5 s. Linka medzi uzlami N4 a N2 vypadne v čase 7 s. Výpadok linky sa vzťahuje ku konkrétnemu rozhraniu a času. Najskôr sa načíta konkrétny uzol z kontajneru `NetDeviceContainer`. Ďalším dôležitým krokom je zvolenie rozhrania na prislúchajúcej linke, ktorá bude deaktivovaná.

Rozhrania sú číslované od 0 po n podľa počtu liniek pripojených k uzlu. Na deaktiváciu linky je použitá funkcia `SetDown`. Nasledujúci kód vypína postupne linku medzi uzlami N3 a N4, N2 a N4. Po výpadku danej linky sú pakety smerované vždy novou trasou, ktorá je prepočítaná protokolom OSPF.

```
NS_LOG_INFO ("Vypadok liniek");

Ptr<Node> n4 = p2pNodes.Get (4);
Ptr<Ipv4> ipv4a = n4->GetObject<Ipv4> ();
uint32_t index1 = 2;
Simulator::Schedule (Seconds (5),&Ipv4::SetDown,ipv4a, index1);

Ptr<Node> n2 = p2pNodes.Get (2);
Ptr<Ipv4> ipv4b = n2->GetObject<Ipv4> ();
uint32_t index2 = 1;
Simulator::Schedule (Seconds (7),&Ipv4::SetDown,ipv4b, index2);
```

Ďalším krokom je nastavenie výstupných súborov pre analyzovanie chovania simulácie. Ako prvé je zaznamenávaný tok paketov do `*.pcap` súborov. Použitá je metóda `EnablePcap`, ktorá pre každé rozhranie point-to-point spojenia vytvorí trasovací súbor s názvom uzlu a číslom rozhrania v zložke `ns-allinone-3.21/ns-3.21/laboratorna_uloha`.

```
NS_LOG_INFO ("Zaznamenanie paketov");

p2p.EnablePcapAll ("laboratorna_uloha/uzol");
p2p.EnableAsciiAll ("laboratorna_uloha/uzol");
```

Druhým výstupom je animácia v grafickej nadstavbe sieťového simulátora NS-3 NetAnim, kde bude zobrazovaný tok paketov sieťou, výpadky v čase 5 s, 7 s a zmeny, ktoré nastali v dôsledku tohoto výpadku. Zobrazené sú názvy uzlov, cieľové a zdrojové adresy. Najskôr je vytvorený objekt `anim` z triedy `AnimationInterface`. Trieda `AnimationInterface` vytvára `*.xml` súbor, ktorý je možné otvoriť v programe NetAnim.

Použité sú funkcie `UpdateNodeDescription` na úpravu názvov uzlov pre lepšiu orientáciu v topológii, `SetConstantPosition`, kvôli namodelovaniu topológie podľa schémy na obrázku Obr. 5.5 a `UpdateNodeColor` na odlíšenie uzlov. Použitá je metóda `EnablePacketMetadata`, ktorá zobrazuje metadáta o paketoch počas simulácie. Varovanie, ktoré sa objaví v príkazovom riadku je možné ignorovať. Spôsobené je nenastavením konštantnej polohy, čo na funkcionalitu laboratórnej úlohy nemá vplyv.

```
NS_LOG_INFO ("Tvorba topologie pre NetAnim");

anim = new AnimationInterface("laboratorna_uloha/animace_OSPF.xml");

anim->UpdateNodeColor(p2pNodes.Get(0), 0, 0, 255);
anim->UpdateNodeDescription(p2pNodes.Get(0), "Server");
anim->SetConstantPosition (p2pNodes.Get(0), -2, 8);

anim->UpdateNodeDescription(p2pNodes.Get(1), "Smerovac_1");
anim->SetConstantPosition (p2pNodes.Get(1), 1.5, 8);

anim->UpdateNodeDescription(p2pNodes.Get(2), "Smerovac_2");
anim->SetConstantPosition (p2pNodes.Get(2), 12.5, 8);

anim->UpdateNodeDescription(p2pNodes.Get(3), "Smerovac_3");
anim->SetConstantPosition (p2pNodes.Get(3), 7, 5);

anim->UpdateNodeDescription(p2pNodes.Get(4), "Smerovac_4");
anim->SetConstantPosition (p2pNodes.Get(4), 9.5, 2);
```

```
anim->UpdateNodeColor(p2pNodes.Get(5), 0, 0, 255);
anim->UpdateNodeDescription(p2pNodes.Get(5), "Klient");
anim->SetConstantPosition (p2pNodes.Get(5), 14, 2);

anim->UpdateNodeDescription(p2pNodes.Get(6), "Smerovac_6");
anim->SetConstantPosition (p2pNodes.Get(6), 7, 11);

anim->UpdateNodeDescription(p2pNodes.Get(7), "Smerovac_7");
anim->SetConstantPosition (p2pNodes.Get(7), 4, 2);
anim->EnablePacketMetadata (true);
```

Ďalej sú generované smerovacie tabuľky v časoch pred výpadkom v čase 2s, po výpadku prvej linky, uzly (N4, N3) v čase 6s a po výpadku druhej linky, uzly (N2, N4) v čase 8s. Smerovacie tabuľky sú generované do zložky `ns-allinone-3.21/ns-3.21/laboratorna_uloha/smerovacie_tabulky.routes`. Na generovanie smerovacích tabuliek je použitá metóda `OutputStreamWrapper`.

```
NS_LOG_INFO ("Generovanie smerovacich tabuliek v casoch zmien");

Ipv4GlobalRoutingHelper smt1;
Ptr<OutputStreamWrapper> routingStream1 = Create<OutputStreamWrapper>
("laboratorna_uloha/smerovacie_tabulky-2s.routes", std::ios::out);
smt1.PrintRoutingTableAllAt (Seconds (2), routingStream1);

Ipv4GlobalRoutingHelper smt2;
Ptr<OutputStreamWrapper> routingStream2 = Create<OutputStreamWrapper>
("laboratorna_uloha/smerovacie_tabulky-6s.routes", std::ios::out);
smt2.PrintRoutingTableAllAt (Seconds (6), routingStream2);

Ipv4GlobalRoutingHelper smt3;
Ptr<OutputStreamWrapper> routingStream3 = Create<OutputStreamWrapper>
("laboratorna_uloha/smerovacie_tabulky-9s.routes", std::ios::out);
smt3.PrintRoutingTableAllAt (Seconds (9), routingStream3);
```

Za generátory smerovacích tabuliek je vložená sonda, ktorá je pripojená k uzlu číslo 3 (N3). Spolu s metódami vkladanými na začiatku skriptu zabezpečuje zber dát a následné vykresľovanie do grafu.

```
Config::Connect("/NodeList/3/$ns3::Ipv4L3Protocol/Tx",MakeCallback(&sent));

for(double time=0; time<endTime; time=time+section)
Simulator::Schedule (Seconds(time), &save, time);
```

Pred funkciou `Run()`, ktorá spúšťa simulátor je vložený modul `FlowMonitor`. Tento modul sleduje tok paketov na zvolených uzloch. Základné parametre, ktoré sú sledované sú počet odoslaných a prijatých paketov, meškanie (delay), jitter a priepustnosť. Modul sa aplikuje pomocou frameworku `FlowMonitor` s použitím `FlowMonitorHelperu`.

```
FlowMonitorHelper flowHelper;
Ptr<FlowMonitor> flowMonitor;
flowMonitor = flowHelper.InstallAll();
Simulator::Stop (Seconds (11));
Simulator::Run ();
```

Na spustenie simulátoru sa využíva funkcia `Simulator::Run()`, ktorá zistí, aké udalosti sú naplánované a vykoná ich. Po funkcii `Run()` je vložená druhá časť kódu pre záznam sledovaných parametrov do konzole.

```
flowMonitor->CheckForLostPackets ();
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>
(flowHelper.GetClassifier ());
std::map<FlowId, FlowMonitor::FlowStats> stats = flowMonitor->GetFlowStats ();
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i =
                        stats.begin (); i != stats.end (); ++i)
        {
Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
NS_LOG_UNCOND( "\n Flow ID: " << i->first << " Src Addr: " << t.sourceAddress << "
Dst Addr: " << t.destinationAddress);
NS_LOG_UNCOND( " Tx Bytes: " << i->second.txBytes);
NS_LOG_UNCOND( " Rx Bytes: " << i->second.rxBytes);
NS_LOG_UNCOND( " Tx Packets: " << i->second.txPackets);
NS_LOG_UNCOND( " Rx Packets: " << i->second.rxPackets);
NS_LOG_UNCOND( " Mean Delay: " << i->second.delaySum.GetSeconds () / (i-
>second.rxPackets) * 1000 << " ms");
NS_LOG_UNCOND( " Mean Jitter: " << i->second.jitterSum.GetSeconds () / (i-
>second.rxPackets - 1) * 1000 << " ms");
NS_LOG_UNCOND( " Throughput: " << i->second.rxBytes * 8.0 / (i-
>second.timeLastRxPacket.GetSeconds () - i->second.timeFirstTxPacket.GetSeconds
()) / 1024 << " Kbps");
        }
flowMonitor->SerializeToXmlFile("OSPF", true, true);
```

Ukončenie simulátoru zabezpečuje funkcia `Simulator::Destroy()`. Pred funkciou `return` je vložený kód, pomocou ktorého je definované, čo bude v grafe vykreslené, ako sa bude volať súbor s grafom, rozsah ôs a ďalšie charakteristiky. Použitá je trieda `Gnuplot`.

```
Simulator::Destroy ();
string fileNameWithNoExtension = "OSPF";
string graphicsFileName = fileNameWithNoExtension + ".png";
string plotFileName = fileNameWithNoExtension + ".plt";
string plotTitle = "Prenosova rychlost na uzle 3";
dataset.SetTitle("Uzol 3");
dataset.SetStyle(Gnuplot2dDataset::LINES_POINTS);
Gnuplot plot(graphicsFileName);
plot.SetTitle(plotTitle);
plot.SetTerminal("png");
plot.SetLegend("Time [s]", "Prenosova rychlost [Mb/s]");
plot.AppendExtra ("set yrange [0:+0.8]");
plot.AddDataset(dataset);
ofstream plotFile(plotFileName.c_str());
plot.GenerateOutput(plotFile);
plotFile.close();

return 0;
}
```

V tejto chvíli je vytvorený skript na simuláciu protokolu OSPF. Po skompilovaní projektu sa v zložke `ns-allinone-3.21/ns-3.21/laboratorna_uloha` sa vytvoria trasovacie súbory `*.pcap` pre každý uzol, súbory `smerovacie_tabulky_xs.routes`, kde `x` je čas kedy bola daná tabuľka generovaná.

Sú to smerovacie tabuľky generované v čase 2 s (pred výpadkom), 6 s po výpadku linky medzi uzlami N3 a N4 a v čase 9 s po výpadku linky medzi uzlami N2 a N4. Ďalším súborom je súbor `animace_OSPF.xml` s uloženými informáciami o simulácii, ktorý sa otvára pomocou programu NetAnim. V konzole je zobrazený počet odoslaných paketov a prijatých paketov, meškanie, jitter a priepustnosť siete medzi dvoma uzlami. Vytvorený je súbor `OSPF.plt`, z ktorého sa generuje graf pomocou programu `gnuplot`.

6.2.1 Spustenie a zobrazenie výsledkov simulácie

Simulácia sa spúšťa tlačidlom Run rovnako ako výpis testovacieho reťazca do konzole na začiatku simulácie. Následne je zavolaná funkcia `Simulator::Run()` a volané metódy `Start()` a `Stop()` pri klientovi a serveri. Po realizácii všetkých udalostí sa simulácia prepne do stavu nečinnosti a zastaví sa aplikácia pre server aj klienta. Nakoniec prebehne vymazanie všetkých objektov simulácie pomocou funkcie `Simulator::Destroy()`.

Pri nastavení premennej `verbose = true` bude prebiehať prenos protokolu TFTP prostredníctvom nespoľahlivého protokolu UDP. Tok paketov je prenášaný z uzlu N5 na uzol N0 (zdrojová adresa 192.168.10.98, cieľová adresa 192.168.10.17). Rýchlosť všetkých liniek je nastavená na 512 kb/s.

V konzole sa zobrazia štatistiky ako na Obr. 6.7. Zo štatistík vidieť, že prenos prebiehal jedným smerom. Pakety neboli potvrdzované po prijatí.

```
Flow ID: 1 Src Addr: 192.168.10.98 Dst Addr: 192.168.10.17
Tx Bytes: 577548
Rx Bytes: 577548
Tx Packets: 549
Rx Packets: 549
Mean Delay: 94.7061 ms
Mean Jitter: 0.174289 ms
Throughput: 495.441 Kbps
```

Obr. 6.7 Prenos pomocou protokolu UDP

Po nastavení premennej `verbose=false`, prebieha prenos pomocou protokolu HTTPS pomocou spoľahlivého protokolu TCP. Pakety budú opäť smerované z uzlu N5 na uzol N0 (zdrojová adresa 192.168.10.98, cieľová adresa 192.168.10.17). V konzole sa zobrazia dva toky dát, keďže TCP protokol čaká na potvrdenie paketov, čo spôsobuje tok dát oboma smermi. Rýchlosť všetkých liniek je nastavená na 512 kb/s. Tok s ID: 1 je tok protokolu HTTPS. Tok s ID: 2 je tok, ktorý potvrdzuje prijaté pakety. Obr. 6.8. Z toho vyplýva že tok s ID: 2 musí byť niekoľkonásobne menší.

```
Flow ID: 1
Tx Bytes: 619380
Rx Bytes: 619380
Tx Packets: 1100
Rx Packets: 1100
Mean Delay: 493.057 ms
Mean Jitter: 7.8951 ms
Throughput: 490.865 Kbps

Flow ID: 2
Tx Bytes: 28656
Rx Bytes: 28656
Tx Packets: 551
Rx Packets: 551
Mean Delay: 3.37545 ms
Mean Jitter: 0.000454553 ms
Throughput: 22.7127 Kbps
```

Obr. 6.8 Prenos paketov prostredníctvom protokolu TCP

Ďalším výstupom sú smerovacie tabuľky v časoch 2s , 6s a 9s. Pre značnú rozsiahlosť smerovacích tabuliek je uvedená časť zo smerovacej tabuľky zachytenej v čase 2s, Obr. 6.9. Smerovacie tabuľky sú rovnaké pri použití UDP aj TCP protokolu.

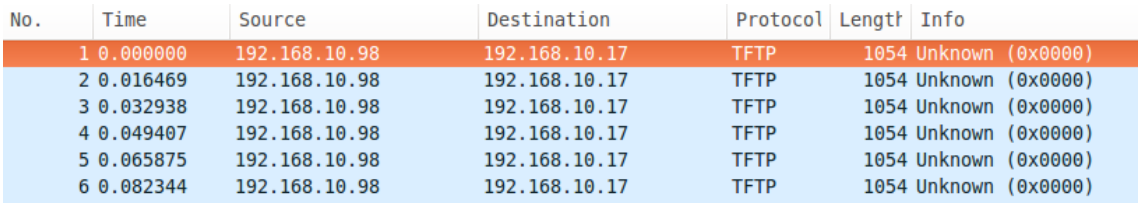
smerovacie_tabulky-2s.routes ×

```
Node: 0 Time: 2s Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Ipv4StaticRouting
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
127.0.0.0       0.0.0.0         255.0.0.0       U     0      -      -   0
192.168.10.16   0.0.0.0         255.255.255.240 U     0      -      -   1
  Priority: -10 Protocol: ns3::Ipv4GlobalRouting
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.10.18   0.0.0.0         UG    -      -      -   1

Node: 1 Time: 2s Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Ipv4StaticRouting
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
127.0.0.0       0.0.0.0         255.0.0.0       U     0      -      -   0
192.168.10.16   0.0.0.0         255.255.255.240 U     0      -      -   1
192.168.10.112  0.0.0.0         255.255.255.240 U     0      -      -   2
192.168.10.0    0.0.0.0         255.255.255.240 U     0      -      -   3
192.168.10.160  0.0.0.0         255.255.255.240 U     0      -      -   4
192.168.10.32   0.0.0.0         255.255.255.240 U     0      -      -   5
  Priority: -10 Protocol: ns3::Ipv4GlobalRouting
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.10.17   192.168.10.17   255.255.255.255 UH    -      -      -   1
192.168.10.114  192.168.10.114  255.255.255.255 UH    -      -      -   2
192.168.10.81   192.168.10.114  255.255.255.255 UH    -      -      -   2
```

Obr. 6.9 Časť smerovacej tabuľky v čase 2 sekundy

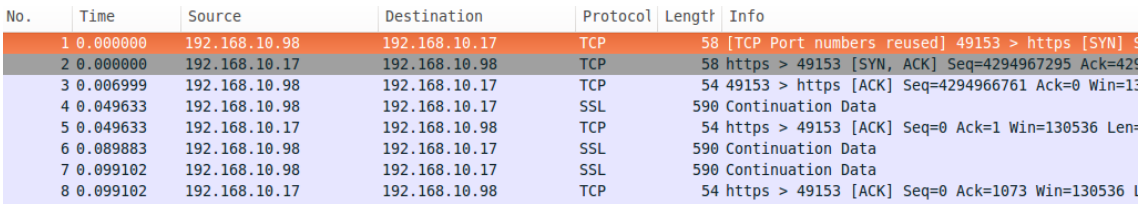
Počas simulácie sú pakety zachytávané do súborov s príponou `*.pcap`. Tieto súbory sú otvárané v sieťovom analyzátore WireShark. Po otvorení súboru `uzol-0-0.pcap`, pri porte nastavenom na hodnotu 69 je výstup nasledujúci ako na Obr. 6.10. Uzol 0 s rozhraním 0 je cieľová destinácia, všetky pakety musia prejsť týmto rozhraním. Pakety nie sú potvrdzované odosielateľovi. Zdrojový port je prvý z dynamických portov 49153. Cieľový port je TFTP. Dĺžka paketu je 1024 bajtov.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.98	192.168.10.17	TFTP	1054	Unknown (0x0000)
2	0.016469	192.168.10.98	192.168.10.17	TFTP	1054	Unknown (0x0000)
3	0.032938	192.168.10.98	192.168.10.17	TFTP	1054	Unknown (0x0000)
4	0.049407	192.168.10.98	192.168.10.17	TFTP	1054	Unknown (0x0000)
5	0.065875	192.168.10.98	192.168.10.17	TFTP	1054	Unknown (0x0000)
6	0.082344	192.168.10.98	192.168.10.17	TFTP	1054	Unknown (0x0000)

Obr. 6.10 Výpis súboru uzol-0-0.pcap pri službe TFTP

Po zmene portu na číslo 443 (HTTPS) prebieha prenos spoľahlivým protokolom TCP v súbore `uzol-0-0.pcap` vidieť výstup, ako na obrázku Obr. 6.11. Zdrojový port je 49153 a cieľový port 443 (služba HTTPS). Jednotlivé pakety sú potvrdzované a prebieha nadviazanie spojenia (three-way handshake). Po prenose všetkých paketov taktiež ukončenie spojenia, ktoré je vidieť na konci relácie. Nazýva sa (four-way handshake). Používajú sa správy SYN, ACK, FIN. Postup ako nadviazanie a ukončenie spojenia v programe Wireshark je popísaný ďalej v texte. Pri HTTPS je používaný protokol SSL (Secure Sockets Layer), ktorý je predchodcom protokolu TLS (Transport Layer Security).



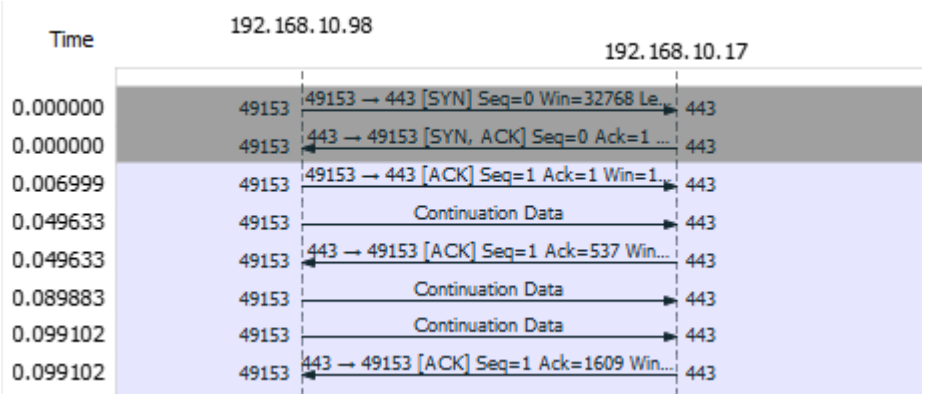
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.98	192.168.10.17	TCP	58	[TCP Port numbers reused] 49153 > https [SYN]
2	0.000000	192.168.10.17	192.168.10.98	TCP	58	https > 49153 [SYN, ACK] Seq=4294967295 Ack=42
3	0.006999	192.168.10.98	192.168.10.17	TCP	54	49153 > https [ACK] Seq=4294966761 Ack=0 Win=1
4	0.049633	192.168.10.98	192.168.10.17	SSL	590	Continuation Data
5	0.049633	192.168.10.17	192.168.10.98	TCP	54	https > 49153 [ACK] Seq=0 Ack=1 Win=130536 Len=
6	0.089883	192.168.10.98	192.168.10.17	SSL	590	Continuation Data
7	0.099102	192.168.10.98	192.168.10.17	SSL	590	Continuation Data
8	0.099102	192.168.10.17	192.168.10.98	TCP	54	https > 49153 [ACK] Seq=0 Ack=1073 Win=130536

Obr. 6.11 Výpis súboru uzol-0-0.pcap pri službe HTTPS

Pri ostatných `*.pcap` súboroch sú zachytené pakety, ktoré cez daný uzol prechádzali, pretože počas simulácie bola trasa niekoľko krát zmenená.

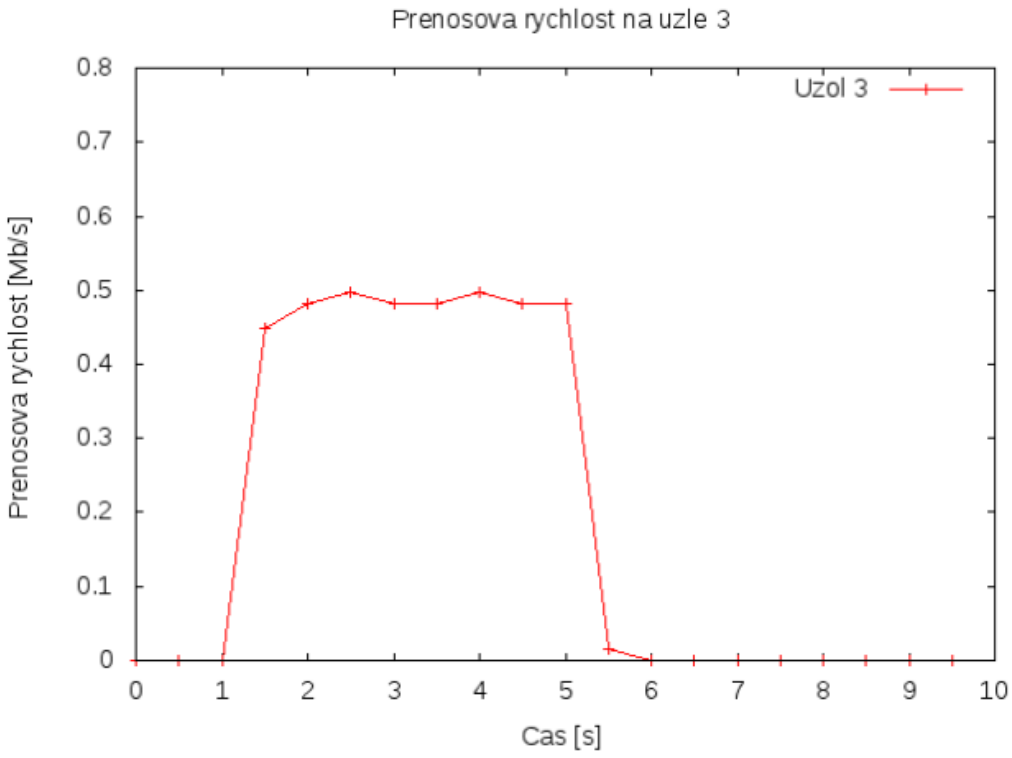
Po otvorení súboru `uzol-0-0.pcap` v sieťovom analyzátore Wireshark je možné popisovaný three-way handshake a four-way handshake vidieť. V záložke `Statistics` kliknutím na položku `Flow Graph` sa otvorí nové okno, v ktorom sa nachádza záznam o tom, ako prebehala komunikácia medzi uzlom N5 a N0.

Na obrázku Obr. 6.12 je zobrazené nadväzovanie spojenia z programu Wireshark so správami [SYN], [SYN, ACK], [ACK]. Podobne sa na konci grafu `(Flow Graph)` nachádza ukončovanie spojenia [FIN, ACK], [ACK], [FIN, ACK], [ACK].



Obr. 6.12 Nadväzovanie spojenia

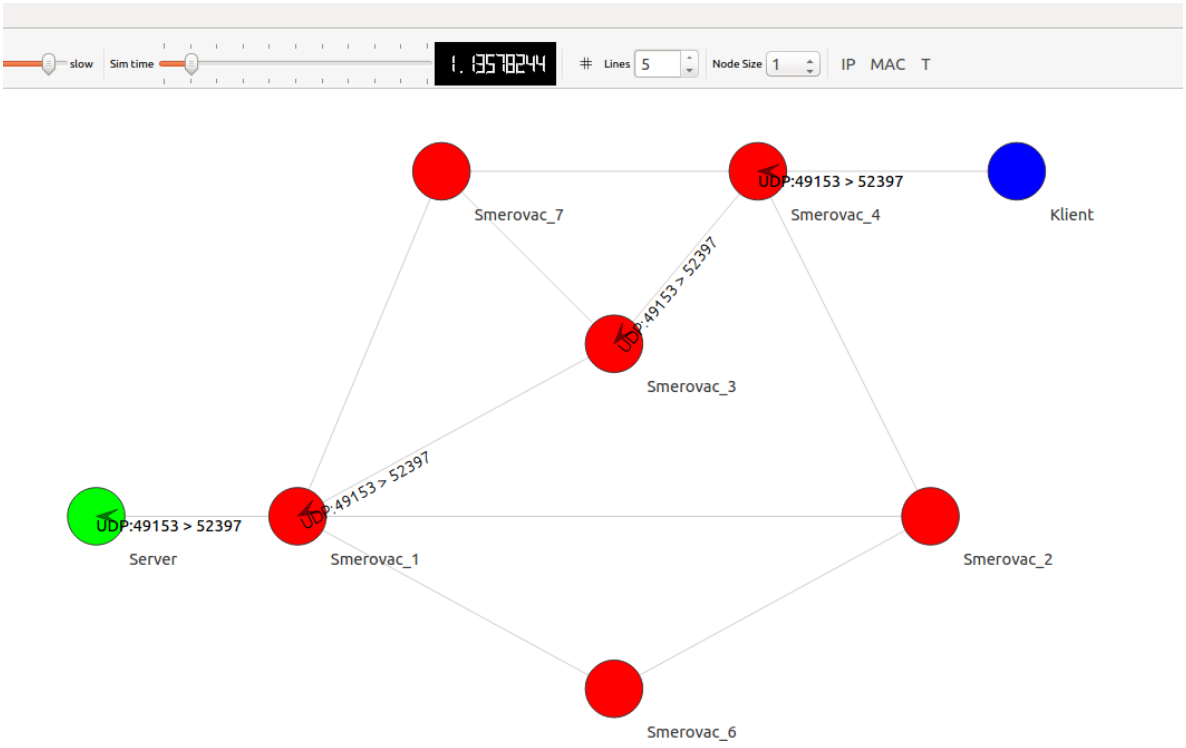
Následne je zobrazený graf toku bitov na uzle 3 (N3). V termináli je nevyhnutné prejsť do zložky, kde sa nachádza súbor `OSPF.plt` zadaním príkazu `cd ns-allinone-3.21/ns-3.21/`. Po prejdení do danej zložky sa graf vytvára zadaním príkazu `gnuplot OSPF.plt`. Týmto príkazom je vygenerovaný obrázok OSPF.png, kde je zobrazená prenosová rýchlosť paketov na uzle N3 v Mbit/s v závislosti na čase. Obr. 6.13. Z grafu je vidieť, že od spustenia aplikácie v čase 1 s idú pakety cez uzol N3 do času 5 s. Po výpadku linky medzi N3 a N4 sa prepočíta nová najvýhodnejšia trasa a pakety nie sú cez tento uzol smerované.



Obr. 6.13 Graf pre uzol N3

Na obrázku Obr. 6.13 je na začiatku grafu v čase 1 s až 2 s vidieť mierny pokles prenosovej rýchlosti paketov na uzle číslo 3 (N3). Tento pokles je spôsobený počiatočným rozširovaním smerovacích tabuliek. Prenosová rýchlosť linky je nastavená na 512 kb/s. Pri začiatku prenosu sa k prenášaným paketom protokolu TFTP alebo HTTPS pridáva prenos protokolu OSPF, ktorý zaťažuje linku a spôsobuje pokles prenosovej rýchlosti paketov.

Súbor `animace_OSPF.xml` sa otvára pomocou programu NetAnim. Server je v tomto prípade uzol 0. Klient je nastavený uzol číslo 5. Pri každom uzle je zobrazený názov. V animácii sú povolené metadáta. Znamená to, že budú zobrazované IP adresy a porty, na ktoré je daný paket smerovaný. Pri protokole UDP bude tok jednosmerný, pri protokole TCP bude nastávať potvrdzovanie paketov - tok dát pôjde oboma smermi. Výsledná topológia je zobrazená na obrázku Obr. 6.14.



Obr. 6.14 Výsledná topológia laboratórnej úlohy

7 ZÁVER

Diplomová práca sa zaoberá problematikou sieťových technológií, protokolov a komunikačných služieb.

V rámci práce bolo zvolené vhodné simulačné prostredie na realizáciu laboratórnych úloh pre výuku sieťových technológií. V kapitole výber simulačného prostredia sú popísané dostupné sieťové simulátory s ich výhodami a nevýhodami. Popísané sú možnosti, ktorými disponujú dané sieťové simulátory ako je generovanie smerovacích tabuliek, trasovacích súborov, animácií, meranie priepustnosti či tvorba grafov. Na základe tejto kapitoly bol zvolený sieťový simulátor NS-3, ktorý bol implementovaný v operačnom systéme Ubuntu v programe Eclipse.

Zo sieťových technológií použitých v laboratórnych úlohách boli pre prvú úlohu zvolené technológie Wi-Fi a Ethernet, pre druhú úlohu bol zvolený smerovací protokol OSPF (Opren Shortest Path First).

Diplomová práca poskytuje podrobný popis skriptu, ktorý vytvára topológiu siete, aplikácie na generovanie a príjem paketov, tvorbu grafov, smerovacích tabuliek, trasovacích súborov štatistík, RTT, zobrazenia pozícií uzlov pri mobilite Wi-Fi staníc a animácií.

Na základe textu diplomovej práce sú vypracované dva laboratórne návody, ktoré obsahujú teoretický úvod k látke, podrobný návod k tvorbe základnej topológie kvôli pochopeniu implementácie konkrétnych technológií použitých v laboratórnych úlohách, podrobne sú popísané zdrojové kódy použité v laboratórnych návodoch a zobrazené sú konkrétne očakávané výstupy. Na základe návodu z prvej časti úlohy je študent schopný vypracovať samostatné úlohy a odpovedať na kontrolné otázky. Laboratórne návody sú preložené do českého jazyka a uvedené v prílohe.

Simulačné prostredie NS-3 môže byť použité k vytvoreniu ďalších laboratórnych úloh pre výuku sieťových technológií kvôli možnostiam implementácie rôznych ďalších protokolov, prepracovanej dokumentácií, podpore vývojárov a širokým možnostiam zobrazovania rôznych štatistík.

LITERATÚRA

[1] *Coding Theory*. Laboratorio Emmy Noether [online]. [cit. 2018-11-20]. Dostupné z: http://ccom.uprrp.edu/~labemmy/?page_id=32

[2] *Circuit switching(CS) vs packet switching(PS) networks | difference between circuit switching and packet switching.* RF Wireless World [online]. [cit. 2018-11-20]. Dostupné z: http://www.rfwireless-world.com/Terminology/circuit-switching-vs-packet-switching.html

[3] *Computer Network | Switching techniques: Message switching.* [online]. [cit. 2018-11-20]. Dostupné z: https://www.geeksforgeeks.org/computer-network-switching-techniques-message-switching/

[4] THAKUR, Dinesh. *Cell Switching (ATM).* Ecomputer notes [online]. [cit. 2018-11-20]. Dostupné z: http://ecomputernotes.com/computernetworkingnotes/switching/cell-switching

[5] *Types of Transmission Media* [online]. [cit. 2018-11-21]. Dostupné z: https://www.geeksforgeeks.org/types-transmission-media/

[6] FILKA, Miloslav. *Přenosová média.* Skriptum VUT.[cit. 2018-11-21].

[7] *Computer network components*. All-about-compiuter-parts.com [online]. [cit. 2018-11-20]. Dostupné z: https://www.all-about-computer-parts.com/computer_network_components.html

[8] *Router*. Technopedia [online]. [cit. 2018-11-21]. Dostupné z: https://www.techopedia.com/definition/2277/router

[9] *How Firewalls Work* [online]. [cit. 2018-11-21]. Dostupné z: https://computer.howstuffworks.com/firewall1.htm

[10] *Gateway.* Technopedia [online]. [cit. 2018-11-21]. Dostupné z: https://www.techopedia.com/definition/5358/gateway

[11] *An Overview of a Personal Area Network* (PAN). Lifewire [online]. [cit. 2018-11-21]. Dostupné z: https://www.lifewire.com/definition-of-pan-817889

[12] *Difference Between LAN, MAN and WAN*. TechDifferences [online]. [cit. 2018-11-21]. Dostupné z: https://techdifferences.com/difference-between-lan-man-and-wan.html

[13] JEŘÁBEK, Jan. *Komunikační technológie*. 2017. Brno.

[14] *Computer Network | Types of area networks – LAN, MAN and WAN* [online]. [cit. 2018-11-21]. Dostupné z: https://www.geeksforgeeks.org/computer-network-types-area-networks-lan-man-wan/

[15] *The OSI Model - Features, Principles and Layers* [online]. [cit. 2018-11-20]. Dostupné z: https://www.studytonight.com/computer-networks/complete-osi-model

[16] SIMONEAU, Paul. *The OSI Model: Understanding the Seven Layers of Computer Networks* [online]. In: . Global knowledge, s. 11 [cit. 2018-11-20]. Dostupné z: http://ru6.cti.gr/bouras-old/WP_Simoneau_OSIModel.pdf

[17] *Network Layer - OSI Model* [online]. [cit. 2018-11-21]. Dostupné z: https://www.studytonight.com/computer-networks/osi-model-network-layer

[18] *Transport Layer* [online]. [cit. 2018-11-21]. Dostupné z: http://www.dcs.bbk.ac.uk/~ptw/teaching/IWT/transport-layer/notes.html

[19] *Application Layer Protocols (DNS, SMTP, POP, FTP, HTTP).* Gradeup [online]. [cit. 2018-11-21]. Dostupné z: https://gradeup.co/application-layer-protocols-dns-smtp-pop-ftp-http-i-ba1194bd-c5ab-11e5-9dcb-5849de73f8e1

[20] JEŘÁBEK, Jan. *Pokročilé komunikační techniky*. 2017. Brno.

[21] *TCP/IP* [online]. [cit. 2018-11-28]. Dostupné z: http://www.ped.muni.cz/wtech/03_studium/teps/teps-03.pdf

[22] KABELOVÁ, Alena a Libor DOSTÁLEK. *Velký průvodce protokoly TCP/IP a systémem DNS*. 5. Brno: Computer Press, 2012.

[23] *IPv4 - Addressing*. Tutorials point [online]. [cit. 2018-11-20]. Dostupné z: https://www.tutorialspoint.com/ipv4/ipv4_addressing.htm

[24] *Network*. IT Base [online]. [cit. 2018-11-20]. Dostupné z: http://itbase.trip.sk/nw/data/protokol/ip_adresa_text.htm

[25] *IPv6 - Overview*. Tutorialspoint [online]. [cit. 2018-11-20]. Dostupné z: https://www.tutorialspoint.com/ipv6/ipv6_overview.htm

[26] *IPv6 (Internet Protocol version 6).* Mendelova univerzita v Brně [online]. [cit. 2018-11-20]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=596

[27] JEŘÁBEK, Jan. *Pokročilé komunikační techniky*. 2017. Brno.

[28] *Transition From IPv4 to IPv6*. Tutorialspoint [online]. [cit. 2018-11-20]. Dostupné z: https://www.tutorialspoint.com/ipv6/ipv6_ipv4_to_ipv6.htm

[29] S. GAST, Mathew. *802.11 Wireless Networks: The Definitive Guide* [online]. Second. 2005 [cit. 2019-04-16]. Dostupné z: https://books.google.cz/books?id=9rHnRzzMHLIC&printsec=frontcover&dq=802.11&hl=sk&sa=X&ved=0ahUKEwi5hc3SqNLhAhVEThoKHYIcDtQQ6AEIMTAB#v=onepage&q&f=false

[30] Wi-Fi - IEEE Standards [online]. [cit. 2018-11-28]. Dostupné z: https://www.tutorialspoint.com/wi-fi/wifi_ieee_standards.htm

[31] *IEEE 802.11ac Gigabit Wi-Fi*. Electronics notes [online]. [cit. 2019-04-16]. Dostupné z: https://www.electronics-notes.com/articles/connectivity/wifi-ieee-802-11/802-11ac.php

[32] *Wi-Fi - Access Protocols*. Tutorials point [online]. [cit. 2019-04-16]. Dostupné z: https://www.tutorialspoint.com/wi-fi/wifi_access_protocols.htm

[33] *What is Ethernet?* [online]. [cit. 2018-12-11]. Dostupné z: https://www.iplocation.net/ethernet

[34] R. FALL, Kevin a Richard W. STEVENS. *TCP/IP Illustrated* [online]. Second. [cit. 2019-04-16]. Dostupné z: https://books.google.cz/books?id=X-l9NX3iemAC&pg=PA999&dq=Ethernet+definition&hl=sk&sa=X&ved=0ahUKEwjNoenR_NHhAhUKCRoKHWApBxcQ6AEIKTAA#v=onepage&q&f=false

[35] *IEEE Standard for Ethernet* [online]. 2016 [cit. 2019-04-16]. Dostupné z: http://www.3utelecom.de/wp-content/uploads/2016/10/802.3-2015_SECTION1.pdf

[36] A. FOROUZAN, Behrouz. *TCP/IP Protocol Suite* [online]. Fourth Edition. 2016 [cit. 2019-04-16]. Dostupné z: http://www.abcd.lk/sliit/Assignment%20Refs/TCP_IP%20Protocol%20Suite%204th%20ed.%20-%20B.%20Forouzan%20(McGraw-Hill,%202010)%20BBS.pdf

[37] *Understanding Network Routing Protocols* [online]. [cit. 2019-04-16]. Dostupné z: https://www.routerfreak.com/understanding-network-routing-protocols/

[38] *Směrovací protokol OSPF* [online]. [cit. 2019-04-16]. Dostupné z: http://www.cs.vsb.cz/grygarek/SPS/lect/OSPF/ospf.html

[39] *Open-Source Network Simulators* [online]. [cit. 2018-11-28]. Dostupné z: http://www.brianlinkletter.com/open-source-network-simulators/

[40] *NetSim Academic* [online]. [cit. 2019-04-16]. Dostupné z: https://www.tetcos.com/netsim-acad.html

[41] *Documentation | ns-3* [online]. [cit. 2019-04-16]. Dostupné z: https://www.nsnam.org/documentation/

[42] *Open-Source Network Simulators* [online]. [cit. 2018-11-28]. Dostupné z: https://www.brianlinkletter.com/psimulator2-graphical-network-simulator/

[43] *Eclipse* [online]. [cit. 2018-11-28]. Dostupné z: https://help.eclipse.org/mars/index.jsp?topic=%2Forg.eclipse.platform.doc.isv%2Fguide%2Fint_eclipse.htm

[44] *Ns-3 Tutorial* [online]. [cit. 2019-04-17]. Dostupné z: https://www.nsnam.org/docs/tutorial/html/

[45] *Ns-3 Doxygen* [online]. [cit. 2019-04-17]. Dostupné z: https://www.nsnam.org/doxygen/

ZOZNAM SKRATIEK

ACK - Acknowledgement

ADT - Advanced Digital Technologies

ARP - Address Resolution Protocol

ATM - Asynchronous Transfer Mode

BGP - Border Gateway Protocol

BSS - Base Service Set

CPU - Central Procesor Unit

CSMA/CA - Carrier Sense Multiple Access with Collision Avoidance

CSMA/CD - Carrier Sense Multiple Access With Collision Detection

DCE - Direct Code Execution

DCF - Distribued Control Function

DHCPv6 - Dynamic Host Control Protocol verzia 6

DNS - Domain Name System

DSL - Digital Subscriber Line

EGP - Exterio Gateway Protocol

EIGRP - Enhanced Interior Gateway Routing Protocol

EPN - Enterprise Area Network

FDDI - Fiber Distribued Data Interface

FTP - File Transfer Protocol

GPL - General Public License

GUI - Graphical User Interface

HTML - HyperText Markup Language

HTTP - Hypertext Transfer Protocol

HTTPS - Hypertext Transfer Protocol Secure

IANA - Internet Assigned Numbers Authority

ICMPv6 - Internet Contorl Management Protocol v6

ICMPv6 - Internet Management Protocol verzia 6

IDE - Integrated Development Environment

IEEE - Institute of Electrical and Electronics Engineers

IGP - Interior Gateway Protocol

IoT - Internet of Things

IP - Internet Protocol

IPv4 - Internet Protocol verzia 4

IPv6 - Internet Protocol verzia 6

IS-IS - Intermediate System to Intermediate System

IS-IS - Intermediate System to Intermediate System

ISO/OSI - Open Systems Interconnection Reference Model

ISP - Internet Service Provider

KVM - Kernel-based Virtual Machine

LAN - Local Area Network

LED - Light Emitting Diode

LLC - Link Logical Control

LTE - Long Term Evolution

LXC - Linux Containers

MAC - Media Access Control

MAN - Metropolitan Area Network

MTU - Maximum transfer Unit

NAT - Network Address Translation

NAT64 - Network Address Translation 64

NAT-PT - Network Address Translation Potr Translation

NS-3 - Network simulator 3

OCL - Object Constraint Language

OFDM - Orthogonal Frequency Division Multiplexing

OSPF - Open Shortest Path Protocol

PAN - Personal Area Network

PPP - Point-to-Point Protocol

QEMU - Quick EMUlator

QoS - Quality of Service

RARP - Reverse Address Resolution Protocol

RIP - Routing Information Protocol

RIPv2 - Routing Information Protocol v2

RJ-45 - Konektor

RTT - Round Trip Time

SDK - Software Development Kit

SIP - Session Initiation Protocol

STP - Shielded Twisted Pair

SYN - Synchronization

SySML - Systems Modeling Language

TCP - Transmission Control Protocol

TCP/IP - Transmission Control Protocol/Internet Protocol

TFTP - Trivial Transfer File Protocol

TLS/SSL - Transport Layer Security/ Secure Sockets Layer

TTL - Time To Live

UDP - User Datagram Protocol

UML - Unified Modeling Language

UTP - Unshielded Twisted Pair

VPN - Virtual Private Network

WAN - Wide Area Network

WiFi - Wireless Fidelity

XML - eXtensible Markup Language

ZOZNAM PRÍLOH

OBSAH PRILOŽENÉHO DVD

Laboratorní úlohy na výuku síťových technologií.pdf - diplomová práca v elektronickej podobe

Laboratorní návod k úloze Technologie Wi-Fi a Ethernet - vypracovaný návod k laboratórnej úlohe Technológia Wi-Fi a Ethernet

Laboratorní návod k úloze Směrovací protokol OSPF - vypracovaný návod k laboratórnej úlohe Smerovací protokol OSPF

OSPF.cc - vypracovanie úlohy Smerovací protokol OSPF s komentovanými časťami kódu zo samostatných úloh (tutoriál k vypracovaniu zadaných samostatných úloh)

OSPF_final.cc - finálne vypracovanie úlohy Smerovací protokol OSPF so samostatnými úlohami

WiFi.cc - vypracovanie úlohy Technológia Wi-Fi a Ethernet s komentovanými časťami kódu zo samostatných úloh (tutoriál k vypracovaniu zadaných samostatných úloh)

Wi-Fi.txt - východzí súbor k úlohe Technológia Wi-Fi a Ethernet

WiFi_final.cc - finálne vypracovanie úlohy Technológia Wi-Fi a Ethernet so samostatnými úlohami

PRÍLOHA A1:

1 TECHNOLOGIE Wi-Fi a ETHERNET

1.1 Teoretický úvod

Laboratorní úloha se zaměřuje na implementaci bezdrátové technologie Wi-Fi a technologie Ethernet v simulačním prostředí NS-3.

Wi-Fi technologie se používá na místech, kde je třeba zpřístupnit síťové služby uživatelům s mobilními koncovými stanicemi. Stanice mezi sebou komunikují pomocí rádiového signálu. Wi-Fi sítě jsou poloduplexního charakteru.

Podle architektury rozlišujeme několik typů Wi-Fi sítí. V laboratorní úloze bude použita infrastrukturní BSS. Součástí infrastrukturního BSS je přístupový bod (Access Point), který je přístupovým bodem k internetu. Přístupový bod vysílá rádiový signál do svého okolí.

Standard IEEE 802.11 pro bezdrátové LAN sítě používá přístupovou technologii CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance), protože všechny stanice vysílají a přijímají na stejném rádiovém kanále. Problémem je, že rádiový kanál nemůže naslouchat, dokud vysílá data a tím pádem není schopen detekovat kolizi jako při Ethernetu. Čím více zařízení se k danému přístupovému bodu připojí, tím nastane více kolizí a rychlost se sníží.

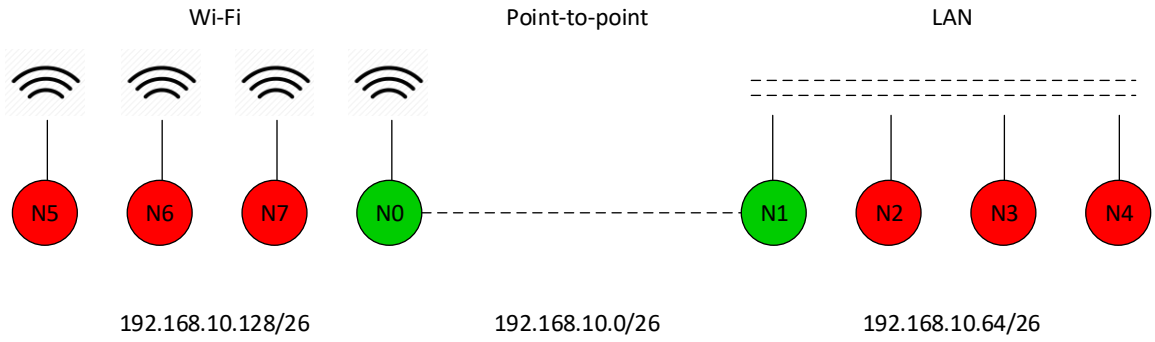
Ethernet je nejrozšířenější technologie používaná v sítích LAN. Standard Ethernet definuje propojovací a signalizační požadavky pro fyzickou vrstvu ve vrstvovém modelu TCP/IP.

Pro řízení přístupu se používá technologie CSMA/CD (Carrier Sense Multiple Access with Collision Detection). Cílem je zabránit ztrátám dat kvůli kolizím, které se předem detekují. Ethernet je definován ve standardu IEEE 802.3. Technika CSMA/CD znamená, že zařízení detekuje, zda se data přenášejí nebo ne. Pokud žádná stanice nevysílá data prostřednictvím přenosového média, tak čekající stanice začne vysílat. Problém vzniká, pokud dvě stanice zjistí, že aktuální žádná ze stanic nevysílá a začnou vysílat současně. Následně detekují kolizi a počkají náhodný čas a začnou vysílat znovu.

1.2 Popis laboratorní úlohy a topologie

Laboratorní úloha se zaměřuje na implementaci bezdrátové technologie Wi-Fi a technologie Ethernet v simulačním prostředí NS-3.

V laboratorní úloze je na začátku vytvořena topologie dle obrázku Obr. 1.1 Počáteční topológie laboratórni úlohy, kde bezdrátová stanice N7 (192.168.10.131/26) pošle požadavek, `Request`, ECHO protokolu na stanici N4 (192.168.10.68/26) komunikující prostřednictvím technologie Ethernet v LAN topologii. Stanice odpoví zprávou odpověď, `Reply`.



Obr. 1.1 Počáteční topológie laboratórni úlohy

1.3 Založení projektu pro práci se simulátorem NS-3

Aby bylo možné vytvářet konkrétní topologii sítě pomocí síťového simulátoru NS-3 implementovaného do vývojového prostředí Eclipse a následně vypracovávat laboratorní úlohy, je nutné dodržet určitý postup práce popsán dále. Projekt má příponu `*.cc` – je psán v programovacím jazyce C ++. Aby projekt fungoval a byla možná jeho kompilace, musí být tento soubor `*.cc` umístěný ve složce `scratch`. V této složce nesmí být najednou více než jeden projekt.

Složka `scratch` se nachází v `/home/nazov_uzivatela/ns-allinone-3.21/ns-3.21`. Původní soubor, "`hello-simulator.cc`", který se v složce `scratch` nachází je třeba smazat. Následně v okně Project Explorer musíme rozbalit složku `ns-3.21` a přejít do složky `scratch` pravým tlačítkem otevřít možnosti a kliknout na `Refresh`. Složka bude prázdná.

Na složku `scratch` klikneme pravým tlačítkem a vybereme `New→File`. Název zvolíme například "`Wi-Fi.cc`". Důležité je uvést příponu, jinak projekt nebude možné zkompilovat. Jelikož potřebujeme projekt zkompilovat, je nutné, aby byl vytvořen jednoduchý funkční program, který vypíše do konzoly textový řetězec. K ověření funkčnosti simulátoru a prvnímu spuštění kompilátoru je třeba vložit následující kód do nově vytvořeného souboru "`Wi-Fi.cc`".

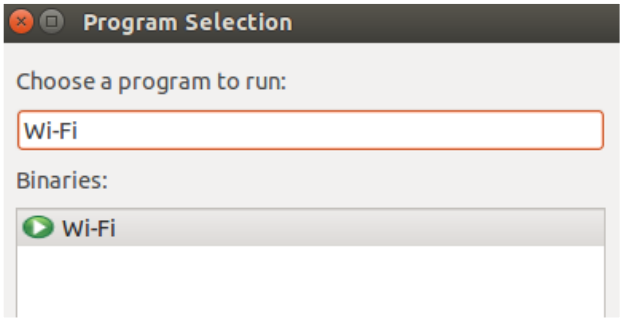
```
#include "ns3/core-module.h"
NS_LOG_COMPONENT_DEFINE ("Wi-Fi");
using namespace ns3;

int main (int argc, char *argv[])
{
     NS_LOG_UNCOND ("Testovaci textovy retazec");
}
```

Dalším krokem je spuštění kompilátoru. Spouští se z `Menu→Run` (zelená šipka). Tímto se vytvoří pomocné soubory potřebné pro debugger. Zkompilované budou všechny projekty, které jsou v NS-3 simulátoru vytvořeny. Kompilace bude trvat několik sekund. V konzole se i přes změnu v programu objeví původní hláška "Hello simulator".

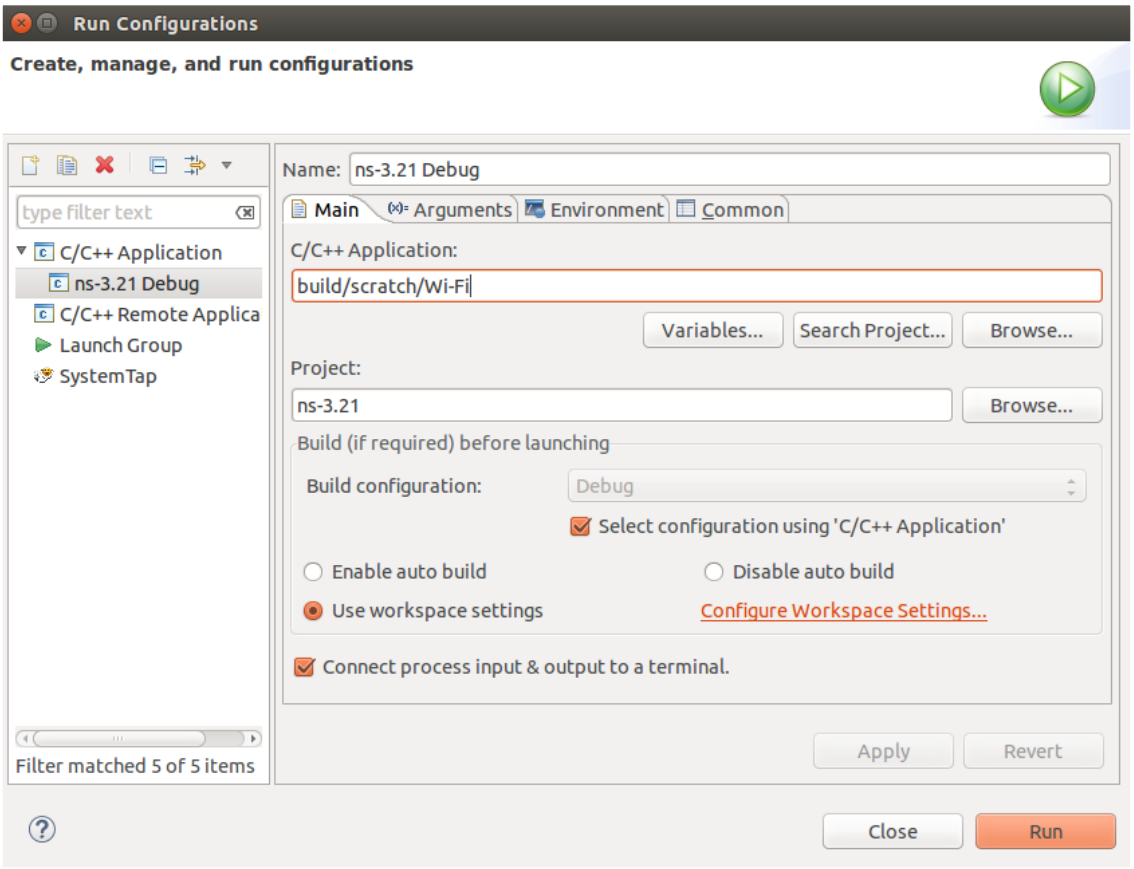
Abychom dostali výpis z nově vytvořeného programu, musíme přejít do nastavení cesty pro debugger. Nastavení realizujeme výběrem položky `Run` (zelená šipka) → `Run Configurations...`.

Po otevření dialogového okna klikneme na ikonu `Search project...`, napíšeme název námi vytvořeného projektu (v tomto případě "`Wi-Fi`"), vybereme tento projekt a zvolíme tlačítkem OK. Zkompilovaný projekt musí mít vedle názvu zelenou šipku, jak je zobrazeno na obrázku Obr. 1.2.



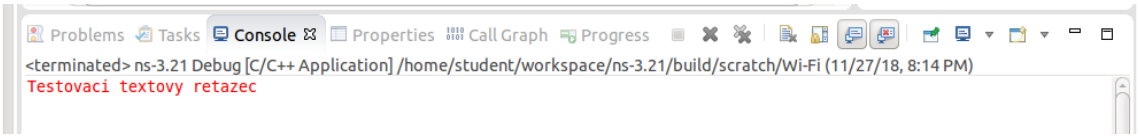
Obr. 1.2 Výběr nově vytvořeného projektu

Výsledná konfigurace je zobrazena na obrázku Obr. 1.3.



Obr. 1.3 Výsledná konfigurace

Po nakonfigurování a spuštění tlačítkem Run se provedou instrukce z programu `"Wi-Fi"` a do konzole se vypíše hláška "Testovaci textovy retezec". Výsledek je zobrazen na obrázku Obr. 1.4.



Obr. 1.4 Výsledek výpisu z konzole

Pokud se do konzoly zobrazí hláška s našeho nově vytvořeného projektu, znamená to, že máme vše nastaveno správně a můžeme přejít k řešení konkrétních úkolů. Před vypracováním úlohy je nutné vymazat celý kód v nově vytvořeném, zkompilovaném souboru a zkopírovat do něj obsah předpřipraveného souboru `Wi-Fi.txt`.

1.4 Tvorba topologie úlohy

Před začátkem laboratorní úlohy vytvoříme složku `laboratorna_uloha` v `/home/ns-allinone-3.21/ns-3.21`. Topologii zobrazenou na obrázku Obr. 1.1, musíme přenést do skriptu v jazyce C ++, který budeme spouštět v simulačním nástroji NS-3. Psaní kódu začíná vložením potřebných modulů, hlavičkových souborů, s příponou `*.h` knihovny NS-3. Moduly zahrneme slovem `#include` a názvem daného hlavičkového souboru do části "`vkládání modulů`". Vzhledem k technologiím použitým v laboratorní úloze budeme muset zahrnout následující moduly:

```
#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/csma-module.h"
#include "ns3/internet-module.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/basic-energy-source.h"
#include "ns3/simple-device-energy-model.h"
#include <iostream>
```

Když máme zahrnuty moduly, musíme definovat jmenný prostor (namespace). Jmenný prostor v NS-3 simulátoru se nazývá `ns3`. Implementací zajišťujeme, že nemusíme zadávat ns3:: operátor před celý kód NS-3 před použitím. Používat budeme také jmenný prostor `std`. Vložíme následující kód:

```
using namespace ns3;
using namespace std;
```

Dále si definujeme funkci, která bude zaznamenávat informace během simulace a v případě chyby zužuje oblast, kde danou chybu budeme hledat. Logování je ve výchozím stavu vypnuto a musí se zapnout. Pro aktivaci logování použijeme následující kód:

```
NS_LOG_COMPONENT_DEFINE ("WiFi");
```

Před funkcí `int main(){}` z důvodu animací pomocí programu NetAnim a zobrazování pozici uzlů (bude vysvětleno později) je nutné vložit následující kód:

```
AnimationInterface * anim = 0;
```

Skript je psán v programovacím jazyce C ++. Na jeho spuštění je třeba použít funkci `int main(){}`, která bude spuštěna jako první. Dále je definována proměnná `verbose`, která nabývá hodnoty `true` nebo `false`. Na jejím základě jsou povoleny nebo zakázány logy, záznamy, aplikace `UdpEchoKlient` a `UdpEchoServer`. Jsou to výpisy zpráv zobrazující přijímány a odesílány pakety. Dále je definován počet stanic CSMA (`nCsma`) a počet Wi-Fi stanic (`nWifi`). Kód vkládáme pod metodu `NodeDistanceAccessPoint()`.

```
int main (int argc, char *argv[])
{
 bool verbose = true;
 uint32_t nCsma = 3;
 uint32_t nWifi = 3;

}
```

Následně pod definici počtu bezdrátových stanic, vkládáme podmínku, kde omezujeme počet možných vložených stanic na 18. Při vyšším počtu by byla simulace výpočetně náročná a nepřehledná. Další blok kódu aktivuje nebo deaktivuje logování.

```
 if (nWifi > 18)
  {
   std::cout << "Pocet uzlov " << nWifi <<
                  " presahuje povoleny pocet" << std::endl;
   exit (1);
  }
```

```
if (verbose)
  {
   LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
   LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
  }
```

Na vytvoření dvou uzlů propojených technologií point-to-point vložíme následující kód. Používáme třídu `NodeContainer`.

```
NodeContainer p2pNodes;
p2pNodes.Create (2);
```

Následně nastavíme parametry linky point-to-point. Bude vytvořena linka s rychlostí 5 Mbps a zpožděním 2 ms. Pro vytvoření takové linky použijeme `PointToPointHelper`. Linku musíme nainstalovat mezi dva uzly.

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

NetDeviceContainer p2pDevices;
p2pDevices = pointToPoint.Install (p2pNodes);
```

Vytvoříme si kontejner uzlů, který bude obsahovat uzly na sběrnici LAN sítě. Tuto LAN síť následně připojíme k jednomu z point-to-point uzlů vybraného z kontejneru uzlů CSMA/CD. Tento uzel bude patřit jak do point-to-point spojení tak bude součástí CSMA/CD LAN sítě.

```
NodeContainer csmaNodes;
csmaNodes.Add (p2pNodes.Get (1));
csmaNodes.Create (nCsma);
```

Podobně jako se nastavovaly parametry point-to-point linky budeme pokračovat s nastavením parametrů LAN uzlů. Parametry se nastavují pomocí `CsmaHelper` voláním funkce `SetChannelAttribute()`. Na konec se parametry musí na danou linku nainstalovat pomocí funkce `Install()`. Popsané kroky odpovídají následujícímu kódu.

```
CsmaHelper csma;
csma.SetChannelAttribute ("DataRate", StringValue ("100Mbps"));
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));

NetDeviceContainer csmaDevices;
csmaDevices = csma.Install (csmaNodes);
```

Následně vytvoříme uzly, které budou tvořit Wi-Fi síť. Uzel N0 bude tvořit

přístupový bod Wi-Fi sítě (Access point). Přístupový bod bude připojen do linky point-to-point. Počet dalších vytvořených uzlů závisí na nastavení na začátku kódu v položce `nWifi`. Momentálně jsou to tři uzly N5, N6 a N7.

```
NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi);
NodeContainer wifiApNode = p2pNodes.Get (0);
```

Následující řádky kódu umožní propojení mezi těmito bezdrátovými uzly a jejich vzájemnou komunikaci. Použijeme na to `YansWifiChannelHelper` a `YansWifiPhyHelper` a nastavíme kanál pro umožnění komunikace. Z kódu vidíme, že se používá výchozí nastavení fyzické vrstvy.

```
YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());
```

Po nastavení fyzické vrstvy nastavíme spojovou vrstvu. Použijeme helper, který nezohledňuje QoS. Na nastavení parametrů tedy použijeme `NqosWifiMacHelper`. Metoda `SetStandard()` umožňuje nastavit právě používaný standard Wi-Fi s různými podporovanými přenosovými rychlostmi. Nastavování pomocí metody `SetRemoteStationManager()` nám umožňuje vybrat, jaký algoritmus pro přenos dat bude použit.

```
WifiHelper wifi = WifiHelper::Default ();
wifi.SetStandard (WIFI_PHY_STANDARD_80211a);
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");

NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();
```

Pokračujeme nastavením SSID (identifikátoru WiFi sítě) a MAC adresy. Vypneme aktivní skenování nastavením "`ActiveProbing`" na hodnotu `false`, čímž aktivujeme pasivní skenování. Pasivní skenování je pomalejší než aktivní, uzel musí čekat na příchod beacon rámce. Beacon rámce jsou posílány periodicky přístupovým bodem (Access Point). Vložením tohoto kódu dokončíme konfiguraci stanic, které nebudou sloužit jako přístupové body. Budou sloužit jako stanice STA s architekturou infrastrukturní BSS.

```
Ssid ssid = Ssid ("VUTBR");
mac.SetType ("ns3::StaWifiMac",
        "Ssid", SsidValue (ssid),
        "ActiveProbing", BooleanValue (false));
```

Po konfiguraci parametrů nainstalujeme nastavení na jednotlivé uzly.

```
NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);
```

Následuje konfigurace přístupového bodu. Výchozí parametry si změníme pomocí `WifiMacHelper`. Následující kód vytvoří MAC vrstvu, která je specifická pro přístupový bod.

```
mac.SetType ("ns3::ApWifiMac",
        "Ssid", SsidValue (ssid));
```

Aby přístupový bod sdílel stejný typ fyzické vrstvy a mohl komunikovat s vytvořenými stanicemi, nainstalujeme mu námi definované parametry.

```
NetDeviceContainer apDevices;
apDevices = wifi.Install (phy, mac, wifiApNode);
```

Přístupový bod musí být stacionární, ale stanice se mohou i nemusí pohybovat. V našem případě budou stanice mobilní. Využijeme k tomu `MobilityHelper`. Pomocí tohoto helperu nastavíme parametry jak přístupovému bodu, tak jednotlivým stanicím. V tomto případě se používá 2D mřížka.

```
MobilityHelper mobility;

mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                    "MinX", DoubleValue (0.0),
                    "MinY", DoubleValue (0.0),
                    "DeltaX", DoubleValue (5.0),
                    "DeltaY", DoubleValue (10.0),
                    "GridWidth", UintegerValue (3),
                    "LayoutType", StringValue ("RowFirst"));
```

Momentálně jsou všechny uzly na staticky umístěny. Na jejich pohyb použijeme funkci `RandomWalkMobilityModel`. Použitím této funkce docílíme, že stanice se budou pohybovat v určitém prostoru definovaném souřadnicemi čtverce. A následně si model nainstalujeme pro všechny STA stanice.

```
mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel",
                           "Bounds", RectangleValue (Rectangle (-50, 50, -50, 50)));
mobility.Install (wifiStaNodes);
```

Abychom dosáhli statickost přístupového bodu, opět použijeme `MobilityHelper` a pomocí něho nastavíme konstantní pozici přístupového bodu a znovu nainstalujeme na potřebný uzel.

```
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (wifiApNode);
```

V tomto okamžiku máme vytvořenou topologii. Abychom mohli přiřazovat IP, adresy a směrovat ve vytvořené topologii musíme nainstalovat internetový protokol. Pro tento účel bude použit `InternetStackHelper`.

```
InternetStackHelper stack;
stack.Install (csmaNodes);
stack.Install (wifiApNode);
stack.Install (wifiStaNodes);
```

Použité budou IPv4 adresy. Pro bezdrátovou síť to bude rozsah 192.168.10.128/26 pro point-to-point linku 192.168.10.0/26 a pro LAN to bude rozsah 192.168.10.64/26. Na přiřazení adres k rozhraním použijeme helper `Ipv4AddressHelper`.

```
Ipv4AddressHelper address;

address.SetBase ("192.168.10.0", "255.255.255.192");
Ipv4InterfaceContainer p2pInterfaces;
p2pInterfaces = address.Assign (p2pDevices);

address.SetBase ("192.168.10.64", "255.255.255.192");
Ipv4InterfaceContainer csmaInterfaces;
csmaInterfaces = address.Assign (csmaDevices);

address.SetBase ("192.168.10.128", "255.255.255.192");
address.Assign (staDevices);
address.Assign (apDevices);
```

Přístupovému bodu nastavíme baterii a pracovní proud. K tomuto účelu využijeme třídu `BasicEnergySource` a `SimpleDeviceEnergyModel`. Nakonec vytvořený model nainstalujeme na konkrétní uzel N0 (přístupový bod).

```
Ptr<BasicEnergySource> energySource = CreateObject<BasicEnergySource>();
Ptr<SimpleDeviceEnergyModel> energyModel =
CreateObject<SimpleDeviceEnergyModel>();
energySource->SetInitialEnergy (700);
energyModel->SetEnergySource (energySource);
energySource->AppendDeviceEnergyModel (energyModel);
energyModel->SetCurrentA (20);

wifiApNode.Get (0)->AggregateObject (energySource);
```

Echo server bude nastaven na uzel číslo N4. Na základě kódu vidíme, že server se bude nacházet na posledním z přidaných uzlů bezdrátový sítě.

```
UdpEchoServerHelper echoServer (7);

ApplicationContainer serverApps = echoServer.Install (csmaNodes.Get (nCsma));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
```

Echo klient se bude nacházet na Wi-Fi stanici N7. Klient se bude nacházet na posledním z přidaných uzlů sítě LAN. Také je nutné vytvořit `ApplicationContainer`, odkud budou pakety posílány.

```
UdpEchoClientHelper echoClient (csmaInterfaces.GetAddress (nCsma), 7);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
```

```
ApplicationContainer clientApps = echoClient.Install (wifiStaNodes.Get (nWifi - 1));
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```

Směrování v síti aktivujeme pomocí `Ipv4GlobalRoutingHelper`, který bude za směrování odpovědný. Vytvoří směrovací tabulky a potřebné náležitosti. `Ipv4GlobalRoutingHelper` má implementován protokol OSPF.

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Dalším nutným krokem je zastavení simulátoru v určitém čase, aby přístupový bod negeneroval beacon rámce donekonečna. Simulace by pak nikdy neskončila. Následujícím řádkem kódu docílíme, že simulace skončí po 10 sekundách, což bude následně vidět v programu NetAnim.

```
Simulator::Stop (Seconds (10.0));
```

Abychom tok paketů zaznamenali do `*.pcap` souborů pro každý uzel a rozhraní použijeme metodu `EnablePcap`, která nám pro každé rozhraní uzlu vytvoří trasovací soubor a umístí jej do složky `/ns-allinone-3.21/ns-3.21/laboratorna_uloha`.

```
pointToPoint.EnablePcapAll ("laboratorna_uloha/WiFi");
phy.EnablePcap ("laboratorna_uloha/WiFi", apDevices.Get (0));
csma.EnablePcap ("laboratorna_uloha/WiFi", csmaDevices.Get (0), true);
```

Následující kód nám vytvoří soubor `Wifi.routes`, kde budou záznamy o směřování – směrovací tabulky. Soubor se bude nacházet ve složce `ns-allinone-3.21/ns-3.21/laboratorna_uloha`. Směrovací tabulky budou zaznamenány v čase 5s. V tomto případě nezávisí od času vytvoření směrovací tabulky, budou stejné po celou dobu simulace. Na jejich generování použijeme `OutputStreamWrapper`.

```
Ipv4GlobalRoutingHelper str;
Ptr<OutputStreamWrapper> routingStream = Create<OutputStreamWrapper>
("laboratorna_uloha/WiFi.routes", std::ios::out);
str.PrintRoutingTableAllAt (Seconds (5), routingStream);
```

Abychom zaznamenávali trajektorii uzlů nastavenou jako náhodný pohyb v určitém přesně vymezeném čtverci, tak vložíme před metodu `NodeDistanceAccessPoint()` následující kód:

```
void
CourseChange (std::string context, Ptr<const MobilityModel> model)
{
  Vector position = model->GetPosition ();
  NS_LOG_UNCOND (context <<
    " x = " << position.x << ", y = " << position.y);
}
```

Následně si tento pohyb budeme moci zobrazit v programu NetAnim a pro jeden konkrétní uzel si můžeme danou trajektorii zobrazit v konzole ve formě `x, y` souřadnic, které se vypisují při změně polohy.

```
std::ostringstream oss;
  oss <<
    "/NodeList/" << wifiStaNodes.Get (nWifi -3)->GetId () <<
    "/$ns3::MobilityModel/CourseChange";

  Config::Connect (oss.str (), MakeCallback (&CourseChange));
```

Na vizualizaci výsledné topologie s tokem paketů použijeme nástroj NetAnim. Před hlavní funkcí `int main(){}` už máme vložený řádek kódu související s animací. Nejdříve je vytvořen objekt anim z `AnimationInterface` a následně určujeme pozici uzlů a přiřazujeme popis, barvu a souřadnice každému z nich. Pro zobrazení informací o posílaných rámcích zprávách a paketech povolíme `PacketMetadata` nastavením hodnoty na `true`.

```
  anim = new AnimationInterface("laboratorna_uloha/animace_WiFi.xml");

  anim->UpdateNodeDescription(wifiStaNodes.Get(0), "N5");
  anim->SetConstantPosition(wifiStaNodes.Get(0), 6, 10);

  anim->UpdateNodeDescription(wifiStaNodes.Get(1), "N6");
  anim->SetConstantPosition(wifiStaNodes.Get(1), 9, 10);

  anim->UpdateNodeDescription(wifiStaNodes.Get(2), "N7");
  anim->SetConstantPosition(wifiStaNodes.Get(2), 12, 10);

  anim->UpdateNodeDescription(p2pNodes.Get(0), "Access point");
  anim->UpdateNodeColor(p2pNodes.Get(0), 0, 0, 255);

  anim->UpdateNodeDescription(csmaNodes.Get(0), "N1_P2P");
  anim->SetConstantPosition (csmaNodes.Get(0), 15, 10);
  anim->UpdateNodeColor(p2pNodes.Get(0), 0, 0, 255);

  anim->UpdateNodeDescription(csmaNodes.Get(1), "N2");
  anim->SetConstantPosition (csmaNodes.Get(1), 22.5, 14);

  anim->UpdateNodeDescription(csmaNodes.Get(2), "N3");
  anim->SetConstantPosition (csmaNodes.Get(2), 22.5, 9);
```

```
anim->UpdateNodeDescription(csmaNodes.Get(3),"N4");
anim->SetConstantPosition (csmaNodes.Get(3), 22.5, 4);

anim->EnablePacketMetadata (true);
```

Před funkci `Run()`, která spouští simulátor vložíme modul `FlowMonitor`. Tento modul sleduje tok paketů na zvolených uzlech. Základní parametry, které můžeme sledovat jsou počet odeslaných a přijatých paketů, zpoždění (delay), jitter a propustnost. Modul aplikujeme pomocí třídy `FlowMonitor` s použitím `FlowMonitorHelperu`.

```
FlowMonitorHelper flowHelper;
Ptr<FlowMonitor> flowMonitor;
flowMonitor = flowHelper.InstallAll(); Simulator::Stop (Seconds (11));
Simulator::Run ();
```

Na spuštění simulátoru využijeme funkci `Simulator::Run()`, která zjistí jaké události jsou naplánovány a provede je. Po funkci `Run()` je vložena druhá část kódu pro záznam parametrů do konzoly a souboru s příponou `*.xml` ve složce `laboratorna_uloha`. Ukončení simulátoru probíhá pomocí funkce `Simulator::Destroy()`.

```
flowMonitor->CheckForLostPackets ();
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>
(flowHelper.GetClassifier ());
std::map<FlowId, FlowMonitor::FlowStats> stats = flowMonitor->GetFlowStats ();
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i =
                    stats.begin (); i != stats.end (); ++i)
  {

Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
  NS_LOG_UNCOND( "\n Flow ID: " << i->first << " Src Addr: " << t.sourceAddress <<
" Dst Addr: " << t.destinationAddress);
  NS_LOG_UNCOND( " Tx Bytes: " << i->second.txBytes);
  NS_LOG_UNCOND( " Rx Bytes: " << i->second.rxBytes);
  NS_LOG_UNCOND( " Tx Packets: " << i->second.txPackets);
  NS_LOG_UNCOND( " Rx Packets: " << i->second.rxPackets);
  NS_LOG_UNCOND( " Mean Delay: " << i->second.delaySum.GetSeconds () / (i-
>second.rxPackets) * 1000 << " ms");
  }
```

```
NS_LOG_UNCOND( " Mean Jitter: " << i->second.jitterSum.GetSeconds () / (i-
>second.rxPackets - 1) * 1000 << " ms");
NS_LOG_UNCOND( " Throughput: " << i->second.rxBytes * 8.0 / (i-
>second.timeLastRxPacket.GetSeconds () - i->second.timeFirstTxPacket.GetSeconds
()) / 1024 << " Kbps");
  }
flowMonitor->SerializeToXmlFile("laboratorna_uloha/uloha1Flowmon", true, true);
Simulator::Destroy ();
//NodeDistanceAccessPoint(-9);
return 0;
}
```

Nakonec vložíme funkci na zobrazování polohy jednotlivých bezdrátových stanic v čase. Následující kód vkládáme před hlavní funkci `int main(){}`.

```
void
CourseChange (std::string context, Ptr<const MobilityModel> model)
{
 Vector position = model->GetPosition ();
 NS_LOG_UNCOND (context <<
  " x = " << position.x << ", y = " << position.y);
}
```

Abychom trajektorii pro libovolný uzel vypsali do konzoly, vložíme následující kód pod generátor směrovacích tabulek.

```
std::ostringstream oss;
oss <<
  "/NodeList/" << wifiStaNodes.Get (nWifi -3)->GetId () <<
  "/$ns3::MobilityModel/CourseChange";

Config::Connect (oss.str (), MakeCallback (&CourseChange));
```

V této chvíli máme hotový skript a můžeme spustit simulaci. Po spuštění se nám zobrazí výpis v konzole, který zobrazuje, jak se pohybovaly jednotlivé stanice a také v jakých časech poslal klient serveru paket a kdy byl paket přijat pro oba směry. Vytvořené budou čtyři soubory `*.pcap`, které lze otevřít v síťovém analyzátoru Wireshark, soubor `animacia_WiFi.xml`, který zobrazuje graficky průběh simulace. Výstupem je také soubor `WiFi.routes`, ve kterém se nacházejí směrovací tabulky. V konzole se objeví informace o počtu přijatých, odeslaných paketů, zpoždění, jitter a propustnost v kb/s.

1.5 Spuštění a zobrazení výsledků

Simulaci spouštíme tlačítkem Run stejně jako výpis testovacího řetězce do konzoly na začátku simulace. Simulace se následně spustí voláním funkce `Simulator::Run()` voláním metod `Start()` a `Stop()` při klientovi a serveru. Po realizaci všech událostí se simulace přepne do stavu nečinnosti a zastaví se aplikace pro server i klienta. Nakonec proběhne vymazání všech objektů simulace pomocí funkce `Simulator::Destroy()`.

V konzole se nám objeví informace o přenesených paketech mezi klientem a serverem s časy, ve kterých byl paket odeslán a přijat. Výpis z konzoly je zobrazen na obrázku Obr. 1.5.

```
At time 2s client sent 1024 bytes to 192.168.10.68 port 7
At time 2.01796s server received 1024 bytes from 192.168.10.131 port 49153
At time 2.01796s server sent 1024 bytes to 192.168.10.131 port 49153
At time 2.03364s client received 1024 bytes from 192.168.10.68 port 7
```

Obr. 1.5 Výpis poslaných a přijatých bajtů z konzole

Dalším výstupem jsou informace o datovém toku mezi jednotlivými uzly. Tok s ID: 1, Obr. 1.7, zobrazuje přenos dat mezi adresami 192.168.10.131/26 a 192.168.10.68/26 (uzel N7 a N4) a tok s ID: 2, Obr. 1.6, zobrazuje opačný tok dat, to znamená mezi adresami 192.168.10.68/26 a 192.168.10.131/26. Ve výpisu vidíme počet přijatých a odeslaných bajtů, počet přijatých a odeslaných paketů, zpoždění (delay), jitter a propustnost linky.

```
Flow ID: 1
Tx Bytes: 1052
Rx Bytes: 1052
Tx Packets: 1
Rx Packets: 1
Mean Delay: 17.955 ms
Mean Jitter: -nan ms
Throughput: 457.74 Kbps
```

Obr. 1.7 Tok dát s ID 1

```
Flow ID: 2
Tx Bytes: 1052
Rx Bytes: 1052
Tx Packets: 1
Rx Packets: 1
Mean Delay: 15.686 ms
Mean Jitter: -nan ms
Throughput: 523.953 Kbps
```

Obr. 1.6 Tok dát s ID 2

Pokud se přesuneme do adresáře `/ns-allinone-3.21/ns-3.21/laboratorna_uloha`, najdeme tam soubor `WiFi.routes`, který zobrazuje směrovací tabulku, která obsahuje seznam cílových adres a jím příslušející výchozí

bránu a rozhraní, na které mají být pakety s danou cílovou adresou posílány. Na obrázku, Obr. 1.8, je jen část z tohoto souboru.

WiFi.routes ×

```
Node: 0 Time: 5s Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Ipv4StaticRouting
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
127.0.0.0       0.0.0.0         255.0.0.0       U     0      -      -   0
192.168.10.0    0.0.0.0         255.255.255.192 U     0      -      -   1
192.168.10.128  0.0.0.0         255.255.255.192 U     0      -      -   2
  Priority: -10 Protocol: ns3::Ipv4GlobalRouting
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.10.2    192.168.10.2    255.255.255.255 UH    -      -      -   1
192.168.10.128  0.0.0.0         255.255.255.192 U     -      -      -   2
192.168.10.64   192.168.10.2    255.255.255.192 UG    -      -      -   1
127.0.0.0       192.168.10.129  255.0.0.0       UG    -      -      -   2
127.0.0.0       192.168.10.130  255.0.0.0       UG    -      -      -   2
127.0.0.0       192.168.10.131  255.0.0.0       UG    -      -      -   2
192.168.10.0    192.168.10.2    255.255.255.192 UG    -      -      -   1
127.0.0.0       192.168.10.2    255.0.0.0       UG    -      -      -   1
```

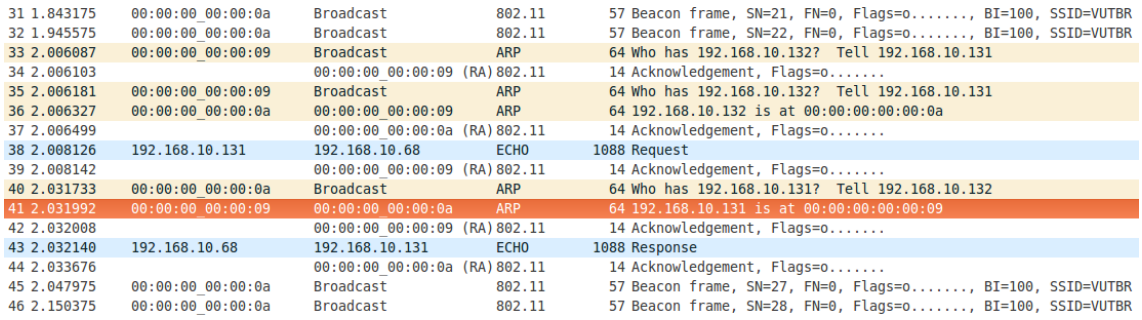
Obr. 1.8 Směrovací tabulka

Také se vytvořily čtyři trasovací soubory pro oba uzly nacházející se ve spojení point-to-point. Přes tyto dva uzly jde celý tok paketů, takže není třeba vypisovat tok z ostatních uzlů. Jsou to soubory `WiFi-0-0.pcap`, `WiFi-0-1.pcap`, `WiFi-1-0.pcap` a `WiFi-1-1.pcap`. V souboru `WiFi-0-0.pcap`, Obr. 1.9, najdeme následující výpis. Použitý protokol je ECHO.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.131	192.168.10.68	ECHO	1054	Request
2	0.018607	192.168.10.68	192.168.10.131	ECHO	1054	Response

Obr. 1.9 Obsah souboru WiFi-0-0.pcap

Na obrázku Obr. 1.10 je zobrazena část ze souboru `WiFi-0-1.pcap`, ve kterém můžeme vidět periodické vysílání beacon rámce přístupovým bodem. Také vidíme použití protokolu ARP (Address Resolution Protocol) na zjištění MAC adresy podle IP adresy a následnou odpověď. Zobrazené je SSID a typ protokolu posílajíci beacon rámce (802.11).



31	1.843175	00:00:00_00:00:0a	Broadcast	802.11	57 Beacon frame, SN=21, FN=0, Flags=o......., BI=100, SSID=VUTBR
32	1.945575	00:00:00_00:00:0a	Broadcast	802.11	57 Beacon frame, SN=22, FN=0, Flags=o......., BI=100, SSID=VUTBR
33	2.006087	00:00:00_00:00:09	Broadcast	ARP	64 Who has 192.168.10.132? Tell 192.168.10.131
34	2.006103		00:00:00_00:00:09 (RA)	802.11	14 Acknowledgement, Flags=o.......
35	2.006181	00:00:00_00:00:09	Broadcast	ARP	64 Who has 192.168.10.132? Tell 192.168.10.131
36	2.006327	00:00:00_00:00:0a	00:00:00_00:00:09	ARP	64 192.168.10.132 is at 00:00:00:00:00:0a
37	2.006499		00:00:00_00:00:0a (RA)	802.11	14 Acknowledgement, Flags=o.......
38	2.008126	192.168.10.131	192.168.10.68	ECHO	1088 Request
39	2.008142		00:00:00_00:00:09 (RA)	802.11	14 Acknowledgement, Flags=o.......
40	2.031733	00:00:00_00:00:0a	Broadcast	ARP	64 Who has 192.168.10.131? Tell 192.168.10.132
41	2.031992	00:00:00_00:00:09	00:00:00_00:00:0a	ARP	64 192.168.10.131 is at 00:00:00:00:00:09
42	2.032008		00:00:00_00:00:09 (RA)	802.11	14 Acknowledgement, Flags=o.......
43	2.032140	192.168.10.68	192.168.10.131	ECHO	1088 Response
44	2.033676		00:00:00_00:00:0a (RA)	802.11	14 Acknowledgement, Flags=o.......
45	2.047975	00:00:00_00:00:0a	Broadcast	802.11	57 Beacon frame, SN=27, FN=0, Flags=o......., BI=100, SSID=VUTBR
46	2.150375	00:00:00_00:00:0a	Broadcast	802.11	57 Beacon frame, SN=28, FN=0, Flags=o......., BI=100, SSID=VUTBR

Obr. 1.10 Obsah souboru WiFi-0-1.pcap

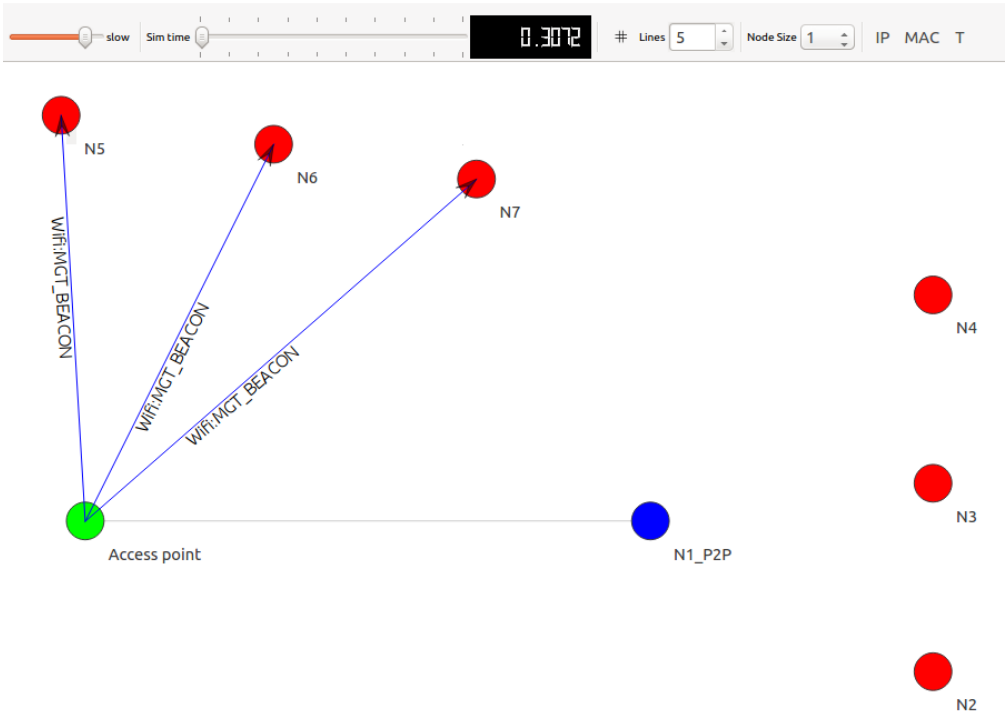
Na závěr otevřeme vygenerovaný soubor `animace_WiFi.xml` v programu NetAnim. Po otevření souboru v programu NetAnim se zobrazí nejdříve bezdrátové stanice s přístupovým bodem a uzly ve spojení point-to-point (N5, N6, N7, Access point, N1_P2P).

V čase 0,25s se zobrazí stanice lokální sítě LAN po asociaci bezdrátových stanic s přístupovým bodem. LAN stanice nemají v programu NetAnim vykreslené linky mezi sebou. Linky mezi nimi existují, avšak NetAnim je nedokáže při LAN a technologií CSMA/CD vykreslit.

Po skončení simulace a přezkoumání toku paketů je vhodné zobrazit trajektorii uzlů, která je vypsána v konzole programu Eclipse. Akci provedeme kliknutím na `Show Properties Tree`, kde při uzlech Node 6, 7, 8 nastavíme `Show Node Trajectory` na `true`.

Před spuštěním simulace je vhodné vypnout mřížku, zobrazující souřadnice a zvětšit topologii lupou kvůli přehlednějšímu zobrazování zpráv, rámců a paketů. V simulaci je vidět všechny toky zobrazené v trasovacích souborech. Přístupový bod je zobrazen zelenou barvou a uzel k němu připojen linkou point-to-point modrou barvou.

Výsledná topologie z programu NetAnim je zobrazena na obrázku Obr. 1.11.



Obr. 1.11 Výslední topológie z programu NetAnim

1.6 Samostatné úkoly

1. Přidejte do topologie jedno další bezdrátové zařízení a jednu stanici do lokální sítě LAN. Bezdrátová stanice bude mít název N8, stanice v lokální síti bude mít název N9. Stanice N9 se bude v programu NetAnim nacházet nad stanicí N4. Stanice N8 se bude nacházet nalevo od přístupového bodu (vhodně upravte souřadnice v programu NetAnim).

2. Po správném přidání uzlů N8 a N9 bude probíhat komunikace mezi nimi, (IP adresy 192.168.10.132/26 a 192.168.10.69/26). Vytvořte další ECHO aplikaci, která bude posílat pakety mezi uzly N4 a N7 (z adresy 192.168.10.131/26 na adresu 192.168.10.68/26). Aplikace na serveru bude začínat v čase 3s a končit v čase 10s. Klientská aplikace bude přijímat pakety v čase 4s a končit v čase 10s. Správnost implementace zkontrolujte v programu Wireshark nebo výpisem z konzoly.

3. Vytvořte aplikaci pro generování paketů protokolu ICMP (ping). K jejímu vytvoření použijeme `OnOffHelper` a `ApplicationContainer`. Na zjednodušenou práci s protokolem ICMP použijeme `V4PingHelper`. Aplikaci vytvoříme vložením následujícího kódu před funkci na aktivaci směrování.

```
InetSocketAddress ca = InetSocketAddress (csmaInterfaces.GetAddress (nCsma-2));
OnOffHelper onoff = OnOffHelper ("ns3::Ipv4RawSocketFactory", ca);
onoff.SetConstantRate (DataRate (15000));
onoff.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer apps = onoff.Install (csmaNodes.Get (nCsma-2));
apps.Start (Seconds (1.0));
apps.Stop (Seconds (10.0));

NS_LOG_INFO ("Create pinger");
V4PingHelper ping = V4PingHelper (csmaInterfaces.GetAddress (nCsma-2));
NodeContainer p;
p.Add (wifiStaNodes.Get (nWifi-3));
apps = ping.Install (p);
apps.Start (Seconds (2.0));
apps.Stop (Seconds (5.0));
```

Ping bude probíhat od Wi-Fi stanice N6 (192.168.10.130/26) na stanici z N3 z LAN (192.168.10.67/26). Upravte aplikaci tak, aby bezdrátová stanice N5 (192.168.10.129/26) pingovala uzel N3 a také uzel z LAN N2 (192.168.10.66/26) pingoval uzel N3. Stanice, které budou ping posílat se přidávají do kontejneru uzlů p. Výpis souboru `WiFi-0-0.pcap` bude vypadat jako na obrázku Obr. 1.12.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.130	192.168.10.67	ICMP	86	Echo (ping) request id=0x0000, seq=0/0, ttl=63
2	0.001955	192.168.10.129	192.168.10.67	ICMP	86	Echo (ping) request id=0x0000, seq=0/0, ttl=63
3	0.004355	192.168.10.67	192.168.10.130	ICMP	86	Echo (ping) reply id=0x0000, seq=0/0, ttl=63
4	0.006247	192.168.10.132	192.168.10.69	ECHO	1054	Request
5	0.006260	192.168.10.67	192.168.10.129	ICMP	86	Echo (ping) reply id=0x0000, seq=0/0, ttl=63
6	0.028854	192.168.10.69	192.168.10.132	ECHO	1054	Response
7	0.995307	192.168.10.129	192.168.10.67	ICMP	86	Echo (ping) request id=0x0000, seq=1/256, ttl=63
8	0.995636	192.168.10.130	192.168.10.67	ICMP	86	Echo (ping) request id=0x0000, seq=1/256, ttl=63
9	0.999612	192.168.10.67	192.168.10.129	ICMP	86	Echo (ping) reply id=0x0000, seq=1/256, ttl=63
10	0.999941	192.168.10.67	192.168.10.130	ICMP	86	Echo (ping) reply id=0x0000, seq=1/256, ttl=63
11	1.995726	192.168.10.129	192.168.10.67	ICMP	86	Echo (ping) request id=0x0000, seq=2/512, ttl=63
12	1.996049	192.168.10.130	192.168.10.67	ICMP	86	Echo (ping) request id=0x0000, seq=2/512, ttl=63
13	2.000031	192.168.10.67	192.168.10.129	ICMP	86	Echo (ping) reply id=0x0000, seq=2/512, ttl=63
14	2.000354	192.168.10.67	192.168.10.130	ICMP	86	Echo (ping) reply id=0x0000, seq=2/512, ttl=63
15	2.004274	192.168.10.131	192.168.10.68	ECHO	1054	Request
16	2.019881	192.168.10.68	192.168.10.131	ECHO	1054	Response

Obr. 1.12 Výpis souboru WiFi-0-0.pcap

Ping mezi stanicemi N2 a N3 je zachycen v souboru `WiFi-1-1.pcap`.

4. Přidejte aplikaci na měření času RTT (Round Trip Time). Na zobrazení obousměrného zpoždění (Round Trip Time) vložíme následující kód před metodu NodeDistanceAccessPoint().

```
static void PingRtt (std::string context, Time rtt)
{
 std::cout << context << " " << rtt << std::endl;
}
```

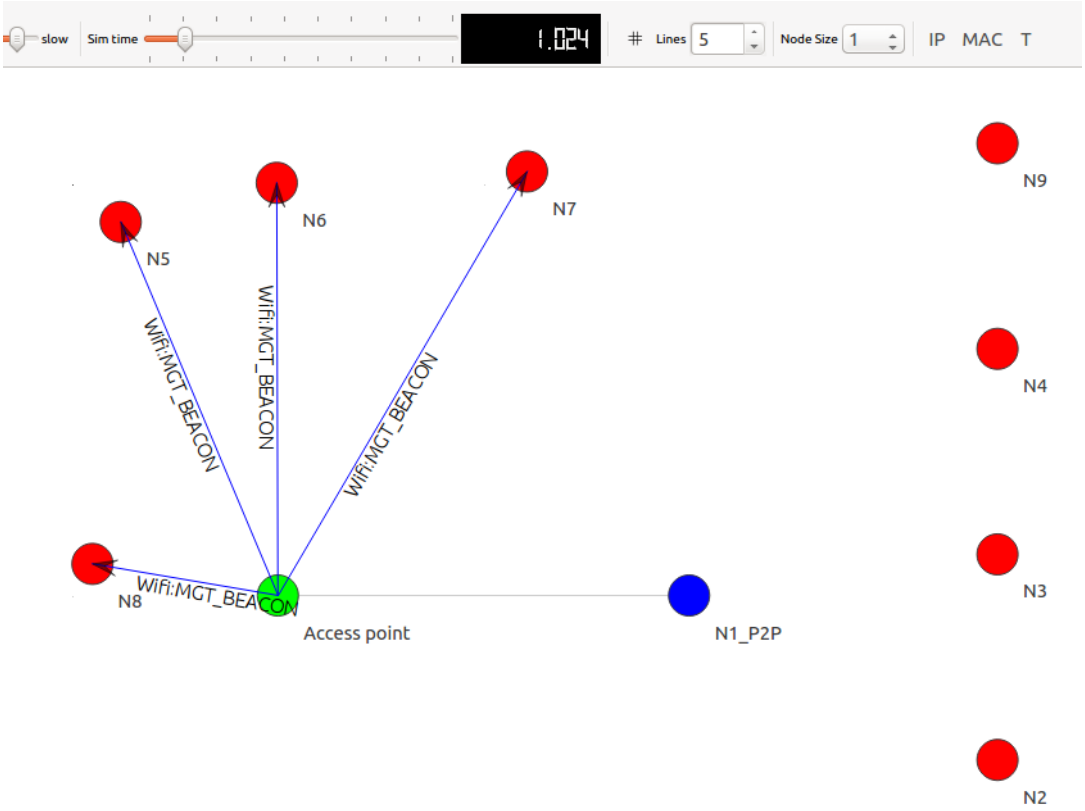
Aby se následně hodnoty RTT vypisovali do konzoly, vložíme následující kód před FlowMonitor. Touto funkcí připojujeme sondu ke všem uzlům a měříme čas RTT.

```
Config::Connect ("/NodeList/*/ApplicationList/*/$ns3::V4Ping/Rtt",MakeCallback
(&PingRtt));
```

Před spuštěním kódu zakomentujte funkci na výpis polohy jednotlivých uzlů do konzoly.

5. Prozkoumejte hodnoty času RTT vypsány v konzoly. Proč je čas RTT pro uzel N2 několikanásobně menší než čas RTT pro uzly N6 a N7?

6. Spusťte soubor `animacia_WiFi.xml` v programu NetAnim. Přehrajte simulaci a sledujte zprávy posílané jednotlivými uzly. Topologie by měla vypadat jako na obrázku Obr. 1.13.



Obr. 1.13 Výslední topológie

7. V programu Wireshark zjistěte SSID sítě a zjistěte podporované rychlosti aktuálně nastaveného Wi-Fi standardu a prozkoumejte zprávu protokolu ARP.

8. Odkomentujte volání metody `NodeDistanceAccessPoint()`, která se nachází pod funkcí `Simulator::Destroy()`. Nastavováním různých vzdáleností uzlů od přístupového bodu zjistěte, který standard má největší dosah v interiéru a který v exteriéru. Jakou má tato vzdálenost přibližnou hodnotu. Zadávané hodnoty jsou v metrech. Výstupy z dané funkce budou vypisovány do konzole. V daném případě se nepoužije mobilita a všechny čtyři uzly budou staticky ve stejné, nastavené, vzdálenosti od přístupového bodu z důvodu demonstrace dosahu signálu v závislosti k použitému standardu.

1.7 Kontrolní otázky

1. Čemu slouží protokol ARP?
2. Co je to SSID a jaké SSID bylo nastaveno v laboratorní úloze?
3. Jaký je rozdíl mezi infrastrukturní architekturou a architekturou ad-hoc?
4. Co znamená Access Point v režimu passive probing?
5. Na jakém portu běží protokol ECHO?
6. Jaký je rozdíl mezi technologií CSMA/CA, CSMA/CD a kde se která používá?
7. Čemu slouží Beacon rámec?

PRÍLOHA A2:

1 SMĚROVACÍ PROTOKOL OSPF

1.1 Teoretický úvod

Hlavním účelem směrovacích protokolů je směrování paketů od zdrojové stanice k cílové stanici. Pakety musí projít mnoha směrovači od zdrojové destinace po cílovou destinaci. O tom, kterým rozhraním bude paket směrován rozhoduje metrika. Metrika se počítá různými způsoby na základě směrovacího protokolu.

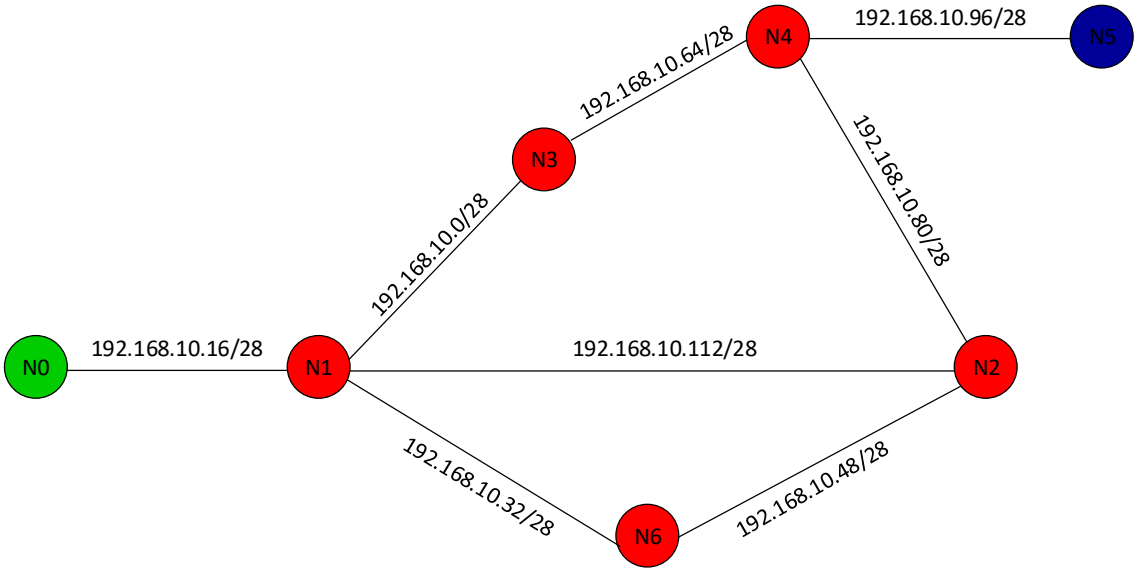
Směrovací protokoly se dělí na link-state, distance-vecor a hybridní. Mezi distance-vector patří protokoly RIP, RIPv2, RIPng. Metrikou je počet přeskoků, což znamená počet směrovačů na cestě k cílové destinaci. Typickým zástupcem link-state směrovacích protokolů je protokol OSPF. Mezi hybridní protokoly patří EIGRP. Je to protokol, který má prvky distance-vector i link-state protokolů.

OSPF (Open Shortest Path First) patří do skupiny link-state protokolů. OSPF posílá všem přímo připojeným sousedům informace o stavu linek. Směrovače posílají hello pakety, které obsahují informace o časovačích, ID směrovače a mnohé další informace. Každý směrovač si udržuje směrovací tabulku, kterou pravidelně aktualizuje a na jejím základě rozesílá příchozí pakety. Čím je metrika menší, tím je cesta preferovanější. Metrika při protokolu OSPF se nazývá cena (cost). Každý OSPF směrovač patří do určité oblasti. Každá oblast má 1 až n směrovačů.

1.2 Popis laboratorní úlohy a topologie

Laboratorní úloha se zabývá směrovacím protokolem OSPF. Postupně je vytvořena topologie sítě, nastavené jsou IP adresy. Vytvořená je aplikace na generování toku paketů. Názorně bude ukázáno, jak se chová protokol OSPF při změnách metriky linek na rozhraních. V laboratorní úloze je použit tok paketů protokolu UDP na portu 52397. Vytvořená bude aplikace generující tok paketů protokolu TFTP a následně aplikace generující tok paketů protokolu HTTPS. V dalším scénáři bude simulovaný výpadek některých linek. Počáteční topologie úlohy je na obrázku Obr. 1.1. Uzel 0

bude serverem, uzel 5 bude klientem. Ostatní uzly budou sloužit jako OSPF směrovače. Adresy jsou voleny tak, aby se každý směrovač nacházel v samostatné síti. Adresy jsou voleny z rozsahu 192.168.10.0/28 (maska 255.255.255.240).



Obr. 1.1 Počáteční topologie laboratórni úlohy

1.3 Založení projektu pro práci se simulátorem NS-3

Aby bylo možný vytvářet konkrétní topologii sítě pomocí síťového simulátoru NS-3 implementovaného do vývojového prostředí Eclipse a následně vypracovávat laboratorní úlohy, je nutné dodržet určitý postup práce popsán dále. Projekt má příponu `*.cc` – je psán v programovacím jazyce C ++. Aby projekt fungoval a byla možná jeho kompilace, musí být tento soubor `*.cc` umístěný ve složce `scratch`. V této složce nesmí být najednou více než jeden projekt jinak by projekt nebyl funkční.

Složka `scratch` se nachází v adresáři `/home/nazov_uzivatela/ns-allinone-3.21/ns-3.21`. Původní soubor, "`hello-simulator.cc`", který se v složce `scratch` nachází je třeba smazat. Následně v okně Project Explorer musíme rozbalit složku `ns-3.21` a přejít do složky `scratch`. Pravým tlačítkem otevřít možnosti a kliknout na `Refresh`. Složka bude prázdná. Na složku

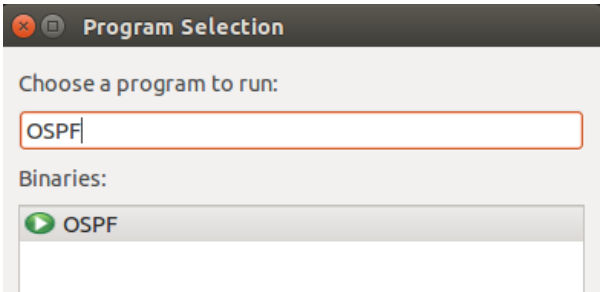
scratch klikneme pravým tlačítkem a vybereme New→File. Název zvolíme například „OSPF.cc“. Důležité je uvést příponu jinak projekt nebude možné zkompilovat. Jelikož potřebujeme projekt zkompilovat je nutné, aby byl vytvořen jednoduchý funkční program, který vypíše do konzole textový řetězec. K ověření funkčnosti simulátoru a první spuštění kompilátoru je třeba vložit následující kód do souboru „OSPF.cc“.

```
#include "ns3/core-module.h"
NS_LOG_COMPONENT_DEFINE ("OSPF");
using namespace ns3;

int main (int argc, char *argv[])
{
     NS_LOG_UNCOND ("Testovaci textovy retazec");
}
```

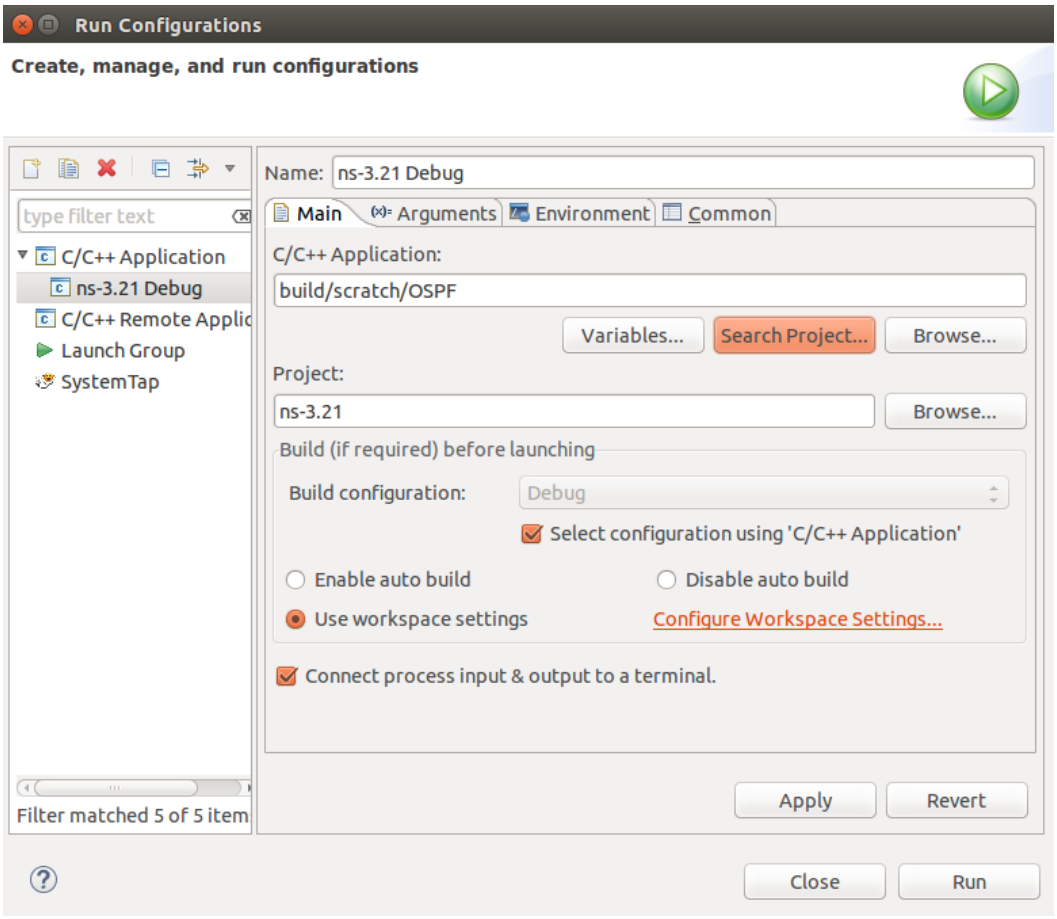
Dalším krokem je spuštění kompilátoru. Spouští se z Menu→Run (zelená šipka). Tímto se vytvoří pomocné soubory potřebné pro debugger. Zkompilované budou všechny projekty, které jsou v NS-3 simulátoru vytvořeny. Kompilace bude trvat několik sekund. V konzole se i přes změnu v programu objeví původní hláška "Hello simulator".

Abychom dostali výpis z nově vytvořeného programu je třeba přejít do nastavení cesty pro debugger. Nastavení realizujeme výběrem položky Run (zelená šipka) → Run Configurations.... Po otevření dialogového okna klikneme na ikonu Search project..., napíšeme název námi vytvořeného projektu (v tomto případě "OSPF"), vybereme tento projekt a zvolíme tlačítkem OK. Zkompilovaný projekt musí mít vedle názvu zelenou šipku, jak je zobrazeno na obrázku Obr. 1.2.



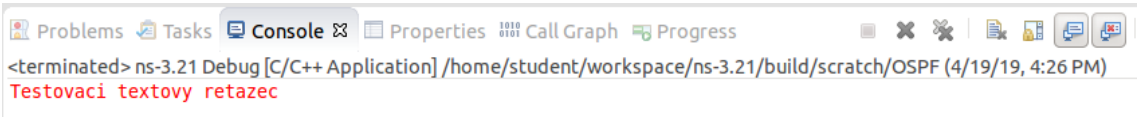
Obr. 1.2 Výběr nově vytvořeného projektu

Výslední konfigurace je zobrazena na obrázku Obr. 1.3.



Obr. 1.3 Ukázka správného nastavení projektu

Po nakonfigurování a spuštění tlačítkem Run se provedou instrukce z programu "`OSPF`" a do konzole se vypíše hláška "Testovaci textovy vypis". Výsledek je zobrazen na obrázku Obr. 1.4.



Obr. 1.4 Ukážka výpisu z konzole

Pokud se v konzole objeví hláška "Testovaci textovy retezec" znamená to, že máme vše nastaveno správně a můžeme přejít k řešení konkrétního úkolu. Před vypracováním úlohy je nutné vymazat celý kód v souboru `OSPF.cc` a vytvořit složku `laboratorne_ulohy v /ns-allinone-3.21/ns-3.21`. Následně můžeme pokračovat dál podle návodu.

1.4 Tvorba topológie úlohy

Topologii zobrazenou na obrázku Obr. 1.1 musíme vepsat do skriptu v jazyce C ++, který budeme spouštět v simulačním nástroji NS-3. Psaní kódu začíná vložením potřebných modulů – hlavičkových souborů s příponou `*.h` knihovny NS-3. Moduly zahrneme slovem `#include` a názvem daného hlavičkového souboru. Vzhledem k technologiím použitým v laboratorní úloze budeme muset zahrnout následující moduly:

```
#include <iostream>
#include <fstream>
#include <string>
#include <cassert>
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/ipv4-global-routing-helper.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor-module.h"
#include "ns3/gnuplot.h"
#include "ns3/stats-module.h"
```

Když máme zahrnuty moduly, musíme deklarovat jmenný prostor (namespace). Jmenný prostor v NS-3 simulátoru se nazývá `ns3`. Po jeho implementaci zajistíme, že nemusíme zadávat ns3:: operátor před celý kód NS-3 před použitím. Také budeme používat jmenný prostor `std`.

```
using namespace ns3;
using namespace std;
```

Dále si definujeme funkci, která bude zaznamenávat informace během simulace a v případě chyby zužuje oblast, kde danou chybu hledat. Logování je ve výchozího stavu vypnuto a musí se zapnout. Pro aktivaci logování použijeme následující kód:

```
NS_LOG_COMPONENT_DEFINE ("OSPF");
```

Před funkcí `int main(){}` z důvodu animací pomocí programu NetAnim a zobrazování pozici uzlů (bude vysvětleno později) a pro potřeby tvorby grafu pomocí programu `gnuplot` je nutné vložit následující kód:

```
AnimationInterface * anim = 0;
Gnuplot2dDataset txAllDataset;
```

Skript píšeme v programovacím jazyce C ++. Na jeho spuštění musíme použít funkci `int main(){}`, která bude spouštěna jako první. Do těla této funkce budeme postupně vkládat celý zdrojový kód s topologií sítě, parametry linek. K definování síťových prvků, IP adres, protokolů budou použity helpery, které usnadňují práci. Nejdříve vložíme funkce na zaznamenávání činnosti simulátoru `LogComponentEnable`.

```
int main (int argc, char *argv[])
{
  #if 0
  LogComponentEnable ("GlobalRoutingHelper", LOG_LOGIC);
  LogComponentEnable ("GlobalRouter", LOG_LOGIC);
  #endif
}
```

Následně vložíme metodu `Ipv4GlobalRouting::RespondTo InterfaceEvent`, jejíž hodnotu nastavíme na `true`. Tato funkce je kritická pro fungování laboratorní úlohy. Pokud by se tato část kódu zakomentovala, protokol OSPF by nebyl schopen reagovat na výpadky linek a jiné změny v topologii během simulace. Po výpadku některé z linek nebo změně metriky by byly pakety zahazovany. Použití téhle funkce je jednou ze dvou možností.

```
Config::SetDefault ("ns3::Ipv4GlobalRouting::RespondToInterfaceEvents",
BooleanValue (true));
```

Zavedená je proměnná `verbose`, která nabývá hodnoty `true` nebo `false`, Proměnná `verbose` je použita na rozhodování podmínky, zda bude použit UDP protokol se službou TFTP nebo TCP protokol se službou HTTPS.

```
bool verbose = (true);
```

Během celého vkládání částí kódu jsou zakomponované komentáře, které se zobrazí v logovacím souboru, aby bylo možné rychleji vyhledat případnou chybu při simulaci. Dále tyto části kódu nebudou popisovány.

```
NS_LOG_UNCOND ("Projekt OSPF");
NS_LOG_INFO ("Vytvorenie uzlu.");
```

Následně budeme vytvářet topologii sítě podle obrázku Obr. 1.1. Začneme vytvořením potřebného počtu uzlů pomocí helperu `NodeContainer` pro spojení point-to-point. Vytvoříme potřebných sedm uzlů. Pokud se vytvoří větší počet simulace bude fungovat, avšak směrovací tabulka bude obsahovat redundantní informace o neexistujících uzlech.

```
NodeContainer p2pNodes;
p2pNodes.Create (7);
```

Když máme vytvořeny jednotlivé uzly, musíme je mezi sebou provázat a vytvořit linky. K tomuto účelu použijeme `NodeContainer`. Pro každou linku vytvoříme vlastní kontejner. Linka mezi uzlem N0 a N1 je ve skriptu označena jako `n0n1` analogicky to platí pro všechny linky. Takovým způsobem bude vytvořeno osm kontejnerů mezi jednotlivými uzly. Po provedení tohoto kódu budou vytvořeny kontejnery dvojic uzlů pro topologii zobrazenou na obrázku Obr. 1.1. Uzly budeme volat pomocí metody `Get()` a jako argument uvedeme číslo uzlu.

```
NodeContainer n0n1 = NodeContainer (p2pNodes.Get (0), p2pNodes.Get (1));
NodeContainer n1n2 = NodeContainer (p2pNodes.Get (1), p2pNodes.Get (2));
NodeContainer n1n3 = NodeContainer (p2pNodes.Get (1), p2pNodes.Get (3));
NodeContainer n1n6 = NodeContainer (p2pNodes.Get (1), p2pNodes.Get (6));
NodeContainer n2n4 = NodeContainer (p2pNodes.Get (2), p2pNodes.Get (4));
NodeContainer n2n6 = NodeContainer (p2pNodes.Get (2), p2pNodes.Get (6));
NodeContainer n3n4 = NodeContainer (p2pNodes.Get (3), p2pNodes.Get (4));
NodeContainer n4n5 = NodeContainer (p2pNodes.Get (4), p2pNodes.Get (5));
```

Dále budeme definovat parametry jednotlivých linek. Použijeme na to `PointToPointHelper`. Všem linkám nadefinujeme rychlost přenosu `Datarate` s hodnotou `512 kb/s` a zpoždění `Delay` na lince `0 ms`.

```
NS_LOG_INFO ("Vytvorenie kanalov.");
PointToPointHelper p2p;
p2p.SetDeviceAttribute ("DataRate", StringValue ("512kbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("0ms"));
```

Po nastavení parametrů linky nainstalujeme tuto hodnotu mezi dvojice uzlů. Pro tento účel použijeme `NetDeviceContainer` a funkci `Install()`. Jako parametr použijeme jednotlivé linky mezi zařízeními z `NodeContaineru`.

```
NetDeviceContainer d0d1 = p2p.Install (n0n1);
NetDeviceContainer d1d2 = p2p.Install (n1n2);
NetDeviceContainer d1d3 = p2p.Install (n1n3);
NetDeviceContainer d1d6 = p2p.Install (n1n6);
NetDeviceContainer d2d4 = p2p.Install (n2n4);
NetDeviceContainer d2d6 = p2p.Install (n2n6);
NetDeviceContainer d3d4 = p2p.Install (n3n4);
NetDeviceContainer d4d5 = p2p.Install (n4n5);
```

Dalším krokem po vytvoření topologie sítě je nastavit sadu protokolů, která bude využívána v práci. Použijeme sadu protokolů TCP/IP. Všechny uzly jsou vytvořeny v kontejneru `p2pNodes`. Protokolovou sadu musíme aplikovat a rozšířit do celé topologie. Tuto funkci nám poskytuje `InternetStackHelper`. Protokolovou sadu nainstalujeme na všechny uzly v topologií (všechny budou fungovat jako směrovače OSPF). Kód bude vypadat následovně:

```
InternetStackHelper internet;
internet.Install (p2pNodes);
```

Adresace bude řešena pomocí helperu `Ipv4AddressHelper`. Z toho vyplývá, že použité adresy budou typu IPv4. Pro každou linku vytvořenou pomocí `NetDeviceContainer` bude přiřazena IPv4 adresa. Adresy byly zvoleny tak, aby každá linka spadala do jiného adresního rozsahu. Jednotlivým rozhraním jsou přiřazeny IPv4 adresy pomocí topology helperu automaticky po zahájení simulace. Nejprve musíme vytvořit pro každou linku `Ipv4InterfaceContainer` a přiřadit mu IP adresu. Kód k popsaným krokem je následující:

```
NS_LOG_INFO ("Priradenie adries");

Ipv4AddressHelper ipv4;
ipv4.SetBase ("192.168.10.16", "255.255.255.240");
Ipv4InterfaceContainer i0i1 = ipv4.Assign (d0d1);
ipv4.SetBase ("192.168.10.112", "255.255.255.240");
Ipv4InterfaceContainer i1i2 = ipv4.Assign (d1d2);
```

```
ipv4.SetBase ("192.168.10.0", "255.255.255.240");
Ipv4InterfaceContainer i1i3 = ipv4.Assign (d1d3);
ipv4.SetBase ("192.168.10.32", "255.255.255.240");
Ipv4InterfaceContainer i1i6 = ipv4.Assign (d1d6);
ipv4.SetBase ("192.168.10.80", "255.255.255.240");
Ipv4InterfaceContainer i2i4 = ipv4.Assign (d2d4);
ipv4.SetBase ("192.168.10.48", "255.255.255.240");
Ipv4InterfaceContainer i2i6 = ipv4.Assign (d2d6);
ipv4.SetBase ("192.168.10.64", "255.255.255.240");
Ipv4InterfaceContainer i3i4 = ipv4.Assign (d3d4);
ipv4.SetBase ("192.168.10.96", "255.255.255.240");
Ipv4InterfaceContainer i4i5 = ipv4.Assign (d4d5);
```

Dalším krokem je nastavování metriky. OSPF protokol udržuje v směrovacích tabulkách informace o metrice cest a následně podle toho, která cesta má nejmenší metriku k cílové destinaci zvolí nejvýhodnější trasu pro přenos paketu. Do logovacího souboru zapíšeme, že nastavujeme metriku. Metriku budeme přiřazovat pomocí funkce `SetMetric(rozhraní, metrika)`. Metriku lze zvolit různou pro každý směr přenosu. Výchozí metrika je v tomto případě 20 pro všechny směry a rozhraní.

```
NS_LOG_INFO ("Nastavenie metriky");

i0i1.SetMetric (0, 20);
i0i1.SetMetric (1, 20);
i1i2.SetMetric (0, 20);
i1i2.SetMetric (1, 20);
i1i3.SetMetric (0, 20);
i1i3.SetMetric (1, 20);
i1i6.SetMetric (0, 20);
i1i6.SetMetric (1, 20);
i2i4.SetMetric (0, 20);
i2i4.SetMetric (1, 20);
i2i6.SetMetric (0, 20);
i2i6.SetMetric (1, 20);
i3i4.SetMetric (0, 20);
i3i4.SetMetric (1, 20);
i4i5.SetMetric (0, 20);
i4i5.SetMetric (1, 20);
```

Aby bylo možné aktivovat směrování mezi různými sítěmi a byl umožněn přechod paketů sítí, použijeme `Ipv4GlobalRoutingHelper`. Daný helper použijeme s výhodou pro danou laboratorní úlohu, protože má implementován protokol OSPF, na základě kterého směřuje pakety v síti. Po jeho aktivaci má celá síť funkcionality protokolu OSPF. Do logovacího souboru zapíšeme, že se aktivovalo směrování. Kód na aktivaci je následující:

```
NS_LOG_INFO ("Aktivacia smerovania");

Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Další krok je vytvoření aplikace, která bude pakety generovat a také aplikace, která bude pakety přijímat. Pro generování paketů byla zvolena funkce `OnOffHelper`. Aplikace na generování paketů bude nastavena na uzel číslo 5 s IP adresou 192.168.10.98/28. Velikost paketu bude 1024 B, cílová destinace bude na rozhraní i0i1, uzel N0 s IP adresou 192.168.10.17/28.

Port je nastaven na náhodné číslo z rozsahu většího než 49151. Použitý transportní protokol bude UDP. V laboratorní úloze bude číslo portu 52397. Proto bude použit `ns3::UdpSocketFactory`.

Změna portu se realizuje na změnou hodnoty v proměnné port: `uint16_t port=52379`. Následně je zavedena proměnná `verbose`, která nabývá hodnoty `true` nebo `false`. Podle této podmínky bude zatím rozhodováno, zda budou pakety generovány nebo ne.

Pro přijímání paketů je zvolen `PacketSinkHelper`. `PacketSinkHelper` a `OnOffHelper` musí mít nastaven stejný port. Pokud vysíláme UDP pakety tak, také musíme UDP pakety očekávat (použití `ns3::UdpSocketFactory`). Platí to i pro `ns3::TcpSocketFactory`. Při vytváření aplikace pro generování i příjem paketů pracujeme se soketem což je kombinace IP adresy a portu. Proměnná `verbose` se nastavuje na začátku skriptu. Pokud je hodnota nastavena na `true`, bude aktivován UDP protokol, pokud bude proměnná `verbose` nastavena na `false`, pakety nebudou generovány. Aplikace pro generátor i příjemce paketů používajících protokol UDP budou vypadat následovně:

```
if(verbose == true){

        NS_LOG_INFO ("Vytvorenie klientskej aplikacie");

        uint16_t port = 52397;

        OnOffHelper onoffUdp ("ns3::UdpSocketFactory", InetSocketAddress
(i0i1.GetAddress (0), port));
        onoffUdp.SetAttribute ("PacketSize", UintegerValue (1024));
        onoffUdp.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
        onoffUdp.SetAttribute("OffTime",
StringValue("ns3::ConstantRandomVariable[Constant=0]"));
        ApplicationContainer apps = onoffUdp.Install (p2pNodes.Get (5));
        apps.Start (Seconds (1.0));
        apps.Stop (Seconds (10.0));

        NS_LOG_INFO ("Vytvorenie aplikacie na strane serveru");

        PacketSinkHelper sinkUdp ("ns3::UdpSocketFactory", Address
(InetSocketAddress (Ipv4Address::GetAny (), port)));
        apps = sinkUdp.Install (p2pNodes.Get (0));
        apps.Start (Seconds (1.0));
        apps.Stop (Seconds (10.0));
 }
```

Dále nastavíme výpadek linky. Linka mezi uzly N3 a N4 vypadne v čase 5 s. Výpadek linky se vztahuje ke konkrétnímu rozhraní a času. Nejdříve se načte konkrétní uzel z kontejneru `NetDeviceContainer`. Dalším důležitým krokem je zvolit rozhraní příslušející lince, kterou chceme deaktivovat.

Rozhraní jsou číslována od 0 po n podle počtu linek připojených k uzlu. Na deaktivaci linky se použije funkce `IPv4::SetDown`. Následující kód vypíná linku mezi uzly N3 a N4 v čase 5 s. Po výpadku jsou pakety směrovány novou trasou, která je přepočítána protokolem OSPF.

```
NS_LOG_INFO ("Vypadok liniek");

Ptr<Node> n4 = p2pNodes.Get (4);
Ptr<Ipv4> ipv4a = n4->GetObject<Ipv4> ();
uint32_t index1 = 2;
Simulator::Schedule (Seconds (5),&Ipv4::SetDown,ipv4a, index1);
```

Dalším postupem je nastavení výstupních souborů pro analyzování chování simulace. Jako první zaznamenáme tok paketů do `*.pcap` souborů. Použijeme metodu `EnablePcap`, která nám pro každé rozhraní point-to-point spojení vytvoří trasovací soubor s názvem uzlu a číslem rozhraní v složce `/ns-allinone-3.21/ns-3.21/laboratorna_uloha`.

```
NS_LOG_INFO ("Zaznamenanie paketov");

p2p.EnablePcapAll ("laboratorna_uloha/uzol");
p2p.EnableAsciiAll ("laboratorna_uloha/uzol");
```

Druhým výstupem bude animace v grafické nadstavbě síťového simulátoru NS-3 Netanim, kde zobrazíme tok paketů sítí, výpadek v době 5 s a změny, které nastaly v důsledku tohoto výpadku. Zobrazené budou názvy uzlů, cílové a zdrojové adresy. Nejdříve bude vytvořen objekt anim z třídy `AnimationInterface`. Třída `AnimationInterface` vytváří `*.xml` soubor, který lze otevřít v programu NetAnim.

Použité budou funkce `UpdateNodeDescription()` pro úpravu názvů uzlů pro lepší orientaci v topologii, `SetConstantPosition()` kvůli namodelování topologie podle schématu na začátku návodu a `UpdateNodeColor()` na odlišení uzlů. Použita je metoda `EnablePacketMetadata()`, která zobrazuje meta informace o paketech během simulace. Varování, které se objeví v příkazovém řádku můžeme ignorovat. Informuje o nenastavení mobility uzlů, což na funkcionalitu laboratorní úlohy nemá vliv. Následující kód vkládáme za kód provádějící záznam paketů do PCAP souborů.

```
NS_LOG_INFO ("Tvorba topologie pre NetAnim");

anim = new AnimationInterface("laboratorna_uloha/animace_OSPF.xml");

anim->UpdateNodeColor(p2pNodes.Get(0), 0, 0, 255);
anim->UpdateNodeDescription(p2pNodes.Get(0), "Server");
anim->SetConstantPosition (p2pNodes.Get(0), -2, 8);

anim->UpdateNodeDescription(p2pNodes.Get(1), "Smerovac_1");
anim->SetConstantPosition (p2pNodes.Get(1), 1.5, 8);
```

```
anim->UpdateNodeDescription(p2pNodes.Get(2), "Smerovac_2");
anim->SetConstantPosition (p2pNodes.Get(2), 12.5, 8);

anim->UpdateNodeDescription(p2pNodes.Get(3), "Smerovac_3");
anim->SetConstantPosition (p2pNodes.Get(3), 7, 5);

anim->UpdateNodeDescription(p2pNodes.Get(4), "Smerovac_4");
anim->SetConstantPosition (p2pNodes.Get(4), 9.5, 2);

anim->UpdateNodeColor(p2pNodes.Get(5), 0, 0, 255);
anim->UpdateNodeDescription(p2pNodes.Get(5), "Klient");
anim->SetConstantPosition (p2pNodes.Get(5), 14, 2);

anim->UpdateNodeDescription(p2pNodes.Get(6), "Smerovac_6");
anim->SetConstantPosition (p2pNodes.Get(6), 7, 11);

anim->EnablePacketMetadata (true);
```

Jako další statistiku si necháme vygenerovat směrovací tabulky v časech před výpadkem v čase 2s, po výpadku první linky a uzly (N4, N3) v čase 6s. Směrovací tabulky budou opět generovány do složky `/ns-allinone-3.21/ns3.21/laboratorna_uloha/smerovacie_tabulky.routes`. Na generování směrovacích tabulek bude použita metoda `OutputStreamWrapper`.

```
NS_LOG_INFO ("Generovanie smerovacich tabuliek v casoch zmien");

Ipv4GlobalRoutingHelper smt1;
Ptr<OutputStreamWrapper> routingStream1 = Create<OutputStreamWrapper>
("laboratorna_uloha/smerovacie_tabulky-2s.routes", std::ios::out);
smt1.PrintRoutingTableAllAt (Seconds (2), routingStream1);

Ipv4GlobalRoutingHelper smt2;
Ptr<OutputStreamWrapper> routingStream2 = Create<OutputStreamWrapper>
("laboratorna_uloha/smerovacie_tabulky-6s.routes", std::ios::out);
smt2.PrintRoutingTableAllAt (Seconds (6), routingStream2);
```

Na zobrazování grafu přenosové rychlosti přes uzel 3 musíme před hlavní funkci vložit metody `sent()`, `save()` a vstupní proměnné.

```
int size = 0;
double startTime = 2.0;
double endTime = 10.0;
double section = 0.5;
```

```
void sent(string path, Ptr<Packet const> packet, Ptr<Ipv4> ipv4,
unsigned int p) {
  if (packet->GetSize() >= 100) {
     size = size + packet->GetSize();
  }
}

void save(double time) {
  dataset.Add(time, (double)size/section*8/(1024*1024));
  size = 0;
}
```

Následně musíme pod generátor směrovacích tabulek vložit kód, který připojuje sondu pro měření počtu přenesených bitů k uzlu číslo 3 a cyklus for(), který bude měnit hodnoty času. Následně se budou ukládat informace pro tvorbu grafu pomocí helperu `GnuplotHelper`. Kód pro definici názvu grafu, popisu os, rozsahu dat bude vložen nakonec po funkci `Simulator::Destroy()`.

```
Config::Connect("/NodeList/3/$ns3::Ipv4L3Protocol/Tx",MakeCallback(&sent));

for(double time=0; time<endTime; time=time+section)
Simulator::Schedule (Seconds(time), &save, time);
```

Před funkci `Run()`, která spouští simulátor vložíme modul `FlowMonitor`. Tento modul sleduje tok paketů na zvolených uzlech. Základní parametry, které můžeme sledovat jsou počet odeslaných a přijatých paketů, zpoždění (delay), jitter a propustnost. Modul aplikujeme pomocí třídy `FlowMonitor` s použitím `FlowMonitorHelperu`.

```
FlowMonitorHelper flowHelper;
Ptr<FlowMonitor> flowMonitor;
flowMonitor = flowHelper.InstallAll();
Simulator::Stop (Seconds (11));
Simulator::Run ();
```

Na spuštění simulátoru využijeme funkci `Simulator::Run()`, který zjistí jaké události jsou naplánovány a provede je. Po funkci `Run()` bude vložena druhá část kódu pro záznam sledovaných parametrů do konzoly.

```
flowMonitor->CheckForLostPackets ();
Ptr<Ipv4FlowClassifier> classifier = DynamicCast<Ipv4FlowClassifier>
(flowHelper.GetClassifier ());
std::map<FlowId, FlowMonitor::FlowStats> stats = flowMonitor->GetFlowStats ();
for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i =
                        stats.begin (); i != stats.end (); ++i)
        {
Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
NS_LOG_UNCOND( "\n Flow ID: " << i->first << " Src Addr: " << t.sourceAddress << "
Dst Addr: " << t.destinationAddress);
NS_LOG_UNCOND( " Tx Bytes: " << i->second.txBytes);
NS_LOG_UNCOND( " Rx Bytes: " << i->second.rxBytes);
NS_LOG_UNCOND( " Tx Packets: " << i->second.txPackets);
NS_LOG_UNCOND( " Rx Packets: " << i->second.rxPackets);
NS_LOG_UNCOND( " Mean Delay: " << i->second.delaySum.GetSeconds () / (i-
>second.rxPackets) * 1000 << " ms");
NS_LOG_UNCOND( " Mean Jitter: " << i->second.jitterSum.GetSeconds () / (i-
>second.rxPackets - 1) * 1000 << " ms");
NS_LOG_UNCOND( " Throughput: " << i->second.rxBytes * 8.0 / (i-
>second.timeLastRxPacket.GetSeconds () - i->second.timeFirstTxPacket.GetSeconds
()) / 1024 << " Kbps");
        }
flowMonitor->SerializeToXmlFile("OSPF", true, true);
```

Ukončení NS-3 simulátoru zajistíme zavoláním funkce `Simulator::Destroy()`. Před funkci `return` vkládáme kód, pomocí kterého definujeme, co bude v grafu vykreslené, jak se bude jmenovat výstupní soubor a také rozsah os a název grafu. Používáme třídu `Gnuplot`.

```
Simulator::Destroy ();
string fileNameWithNoExtension = "OSPF";
string graphicsFileName = fileNameWithNoExtension + ".png";
string plotFileName = fileNameWithNoExtension + ".plt";
string plotTitle = "Prenosova rychlost na uzle 3";
dataset.SetTitle("Uzol 3");
dataset.SetStyle(Gnuplot2dDataset::LINES_POINTS);
Gnuplot plot(graphicsFileName);
plot.SetTitle(plotTitle);
plot.SetTerminal("png");
plot.SetLegend("Time [s]", "Prenosova rychlost [Mb/s]");
```

```
plot.AppendExtra ("set yrange [0:+0.8]");
        plot.AddDataset(dataset);
        ofstream plotFile(plotFileName.c_str());

        plot.GenerateOutput(plotFile);
        plotFile.close();

        return 0;
}
```

V této chvíli máme hotový skript a můžeme spustit simulaci. Po zkompilovaný projektu se v složce `ns-allinone-3.21/ns-3.21/laboratorna_uloha` se vytvoří trasovací soubory `*.pcap` pro každý uzel, soubory `smerovacie_tabulky_xs.routes`, kde `x` je čas, kdy byla daná tabulka generována.

Jsou to směrovací tabulky generované v čase 2s (před výpadkem) a 6s (po výpadku) linky mezi uzly N3 a N4. Dalším souborem bude soubor `animace_OSPF.xml` s uloženými informacemi o simulaci, který otevřeme pomocí programu NetAnim. V konzoly bude vidět počet odeslaných paketů a přijatých paketů, zpoždění, jitter a propustnost sítě mezi dvěma uzly. Vytvořený je soubor `OSPF.plt`, který budeme otevírat pomocí programu gnuplot.

1.5 Spuštění a zobrazení výsledků simulace

Simulaci spouštíme tlačítkem Run stejně jako výpis testovacího řetězce do konzole na začátku simulace. Simulace se následně spustí voláním funkce `Simulator::Run()` voláním metod `Start()` a `Stop()` při klientovi a serveru. Po realizaci všech událostí se simulace přepne do stavu nečinnosti a zastaví se aplikace pro server i klienta. Nakonec proběhne vymazání všech objektů simulace pomocí funkce `Simulator::Destroy()`.

Pakety budou přenášeny z uzlu N5 na uzel N0 (zdrojová adresa 192.168.10.98/28, cílová adresa 192.168.10.17/28). Rychlost všech linek je nastavena na 512 kb/s.

V konzoly se zobrazí statistiky jako na Obr. 1.5. Ze statistik vidíme, že přenos probíhal jedním směrem. Pakety nebyly potvrzovány po přijetí.

```
Flow ID: 1 Src Addr: 192.168.10.98 Dst Addr: 192.168.10.17
Tx Bytes: 577548
Rx Bytes: 577548
Tx Packets: 549
Rx Packets: 549
Mean Delay: 94.7061 ms
Mean Jitter: 0.174289 ms
Throughput: 495.441 Kbps
```

Obr. 1.5 Prenos pomocí protokolu UDP

Dalším výstupem jsou směrovací tabulky v časech 2s a 6s. Pro značnou rozsáhlost směrovacích tabulek je uvedena jenom část ze směrovací tabulky zachycené v čase 2s, Obr. 1.6. V tabulce vidíme cílové adresy a výchozí bránu, na kterou mají být pakety s danou adresou směrovány.

smerovacie_tabulky-2s.routes

```
Node: 0 Time: 2s Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Ipv4StaticRouting
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
127.0.0.0       0.0.0.0         255.0.0.0       U     0      -      -   0
192.168.10.16   0.0.0.0         255.255.255.240 U     0      -      -   1
  Priority: -10 Protocol: ns3::Ipv4GlobalRouting
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.10.18   0.0.0.0         UG    -      -      -   1

Node: 1 Time: 2s Ipv4ListRouting table
  Priority: 0 Protocol: ns3::Ipv4StaticRouting
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
127.0.0.0       0.0.0.0         255.0.0.0       U     0      -      -   0
192.168.10.16   0.0.0.0         255.255.255.240 U     0      -      -   1
192.168.10.112  0.0.0.0         255.255.255.240 U     0      -      -   2
192.168.10.0    0.0.0.0         255.255.255.240 U     0      -      -   3
192.168.10.32   0.0.0.0         255.255.255.240 U     0      -      -   4
  Priority: -10 Protocol: ns3::Ipv4GlobalRouting
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.10.17   192.168.10.17   255.255.255.255 UH    -      -      -   1
192.168.10.114  192.168.10.114  255.255.255.255 UH    -      -      -   2
192.168.10.81   192.168.10.114  255.255.255.255 UH    -      -      -   2
192.168.10.49   192.168.10.114  255.255.255.255 UH    -      -      -   2
192.168.10.2    192.168.10.2    255.255.255.255 UH    -      -      -   3
```

Obr. 1.6 Část směrovací tabulky v čase 2s

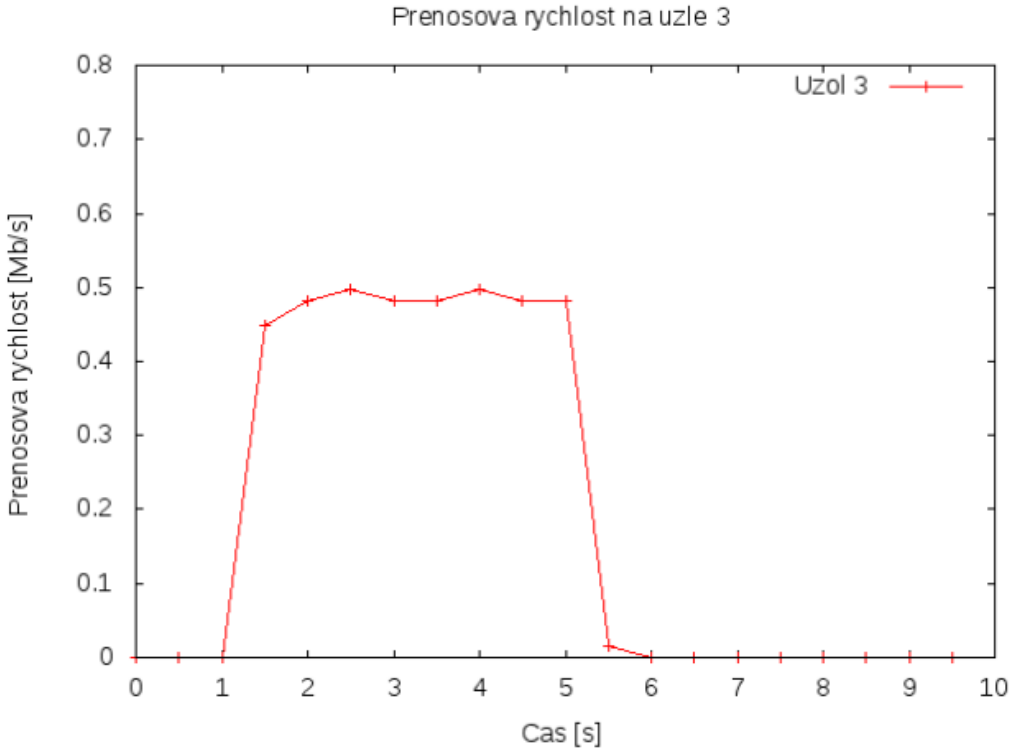
Během simulace se zachycovaly pakety do souborů s příponou `*.pcap`. Tyto soubory můžeme otevřít v síťovém analyzátoru Wireshark. Pokud otevřeme soubor `uzol-0-0.pcap`, při portu nastaveném na hodnotu 52397 uvidíme následující

výstup jako na Obr. 1.7. Uzel 0 s rozhraním 0 je cílová destinace, všechny pakety musí projít tímto rozhraním. Pakety nejsou potvrzovány odesílateli. Jak můžeme vidět, zdrojový port je první s dynamických portů 49153. Cílový port je 52397. Délka paketu je 1024 bajtů.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.98	192.168.10.17	UDP	1054	Source port: 49153 Destination port: 52397
2	0.016469	192.168.10.98	192.168.10.17	UDP	1054	Source port: 49153 Destination port: 52397
3	0.032938	192.168.10.98	192.168.10.17	UDP	1054	Source port: 49153 Destination port: 52397
4	0.049407	192.168.10.98	192.168.10.17	UDP	1054	Source port: 49153 Destination port: 52397
5	0.065875	192.168.10.98	192.168.10.17	UDP	1054	Source port: 49153 Destination port: 52397
6	0.082344	192.168.10.98	192.168.10.17	UDP	1054	Source port: 49153 Destination port: 52397
7	0.098813	192.168.10.98	192.168.10.17	UDP	1054	Source port: 49153 Destination port: 52397
8	0.115282	192.168.10.98	192.168.10.17	UDP	1054	Source port: 49153 Destination port: 52397

Obr. 1.7 Výpis souboru uzol-0-0.pcap při portu nastaveném na 52397

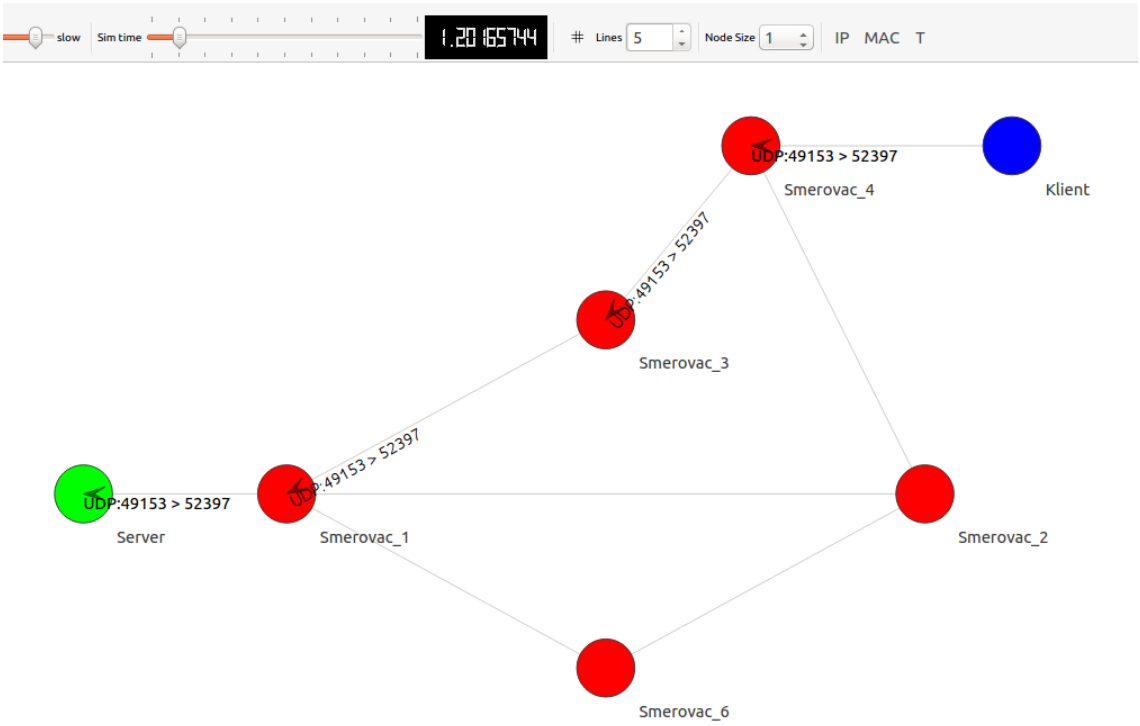
Dále si zobrazíme graf toku bitů na uzlu 3 (N3). V terminálu přejdeme do složky, kde se nachází soubor `OSPF.plt`. Do složky se dostaneme zadáním příkazu `cd ns-allinone-3.21/ns-3.21/`. Ve složce zadáme příkaz `gnuplot OSPF.plt`. Tímto příkazem vygenerujeme obrázek OSPF.png, kde bude zobrazena přenosová rychlost paketů na uzlu číslo 3 (N3) v Mbit/s v závislosti na čase, Obr. 1.8. Z grafu je vidět, že od spuštění aplikace v čase 1 s jdou pakety přes uzel do času 5 s. Po výpadku linky mezi uzly N3 a N4 se přepočítá výhodnější trasa a pakety nejsou přes uzel N3 směrovány.



Obr. 1.8 Graf toku pro uzel N3

Na začátku v čase 1 s až 2 s, Obr. 1.8, je přenosová rychlost paketů nižší, protože probíhá rozšiřovaní směrovacích tabulek protokolem OSPF, které zatěžuje přenosový kanál.

Pomocí programu NetAnim otevřeme soubor `animace_OSPF.xml`. Příjemce paketů je v tomto případě uzel 0. Jako odesílatel je nastaven uzel číslo 5. Při každém uzlu je zobrazen název. V animaci jsou povoleny meta informace. Znamená to, že budou zobrazovány IP adresy a porty, na které je daný paket směrován. Při protokolu UDP bude tok jednosměrný. Výsledná topologie je zobrazena na obrázku Obr. 1.9.



Obr. 1.9 Výslední topológie laboratórni úlohy

V pakety budou směrovány prostřednictvím směrovače 4-> 3-> 1. V době 5 s nastane výpadek linky mezi směrovači 3 a 4. Pakety budou směrovány prostřednictvím směrovače 4-> 2-> 1.

1.6 Samostatné úkoly

1. Upravte metriku tak, aby po výpadku linky v době 5s byly pakety směrovány prostřednictvím směrovačů 4-> 2-> 6-> 1.

2. Změňte UDP aplikaci tak, aby generovala pakety TFTP (změny si všimněte v síťovém analyzátoru Wireshark).

3. Vytvořte aplikaci používající protokol TCP. Typ služby bude HTTPS. Aplikace TCP se aktivuje při zadání hodnoty `bool verbose=false` (upravte podmínku). Použitý bude `ns3::TcpSocketFactory`. Po správném vytvoření aplikace a nastavení hodnoty `verbose=false` vidíme v souboru `uzol-0-0.pcap` následující výstup jako na obrázku Obr. 1.10.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.10.98	192.168.10.17	TCP	58	[TCP Port numbers reused] 49153 > https [SYN]
2	0.000000	192.168.10.17	192.168.10.98	TCP	58	https > 49153 [SYN, ACK] Seq=4294967295 Ack=42
3	0.006999	192.168.10.98	192.168.10.17	TCP	54	49153 > https [ACK] Seq=4294966761 Ack=0 Win=1
4	0.049633	192.168.10.98	192.168.10.17	SSL	590	Continuation Data
5	0.049633	192.168.10.17	192.168.10.98	TCP	54	https > 49153 [ACK] Seq=0 Ack=1 Win=130536 Len
6	0.089883	192.168.10.98	192.168.10.17	SSL	590	Continuation Data
7	0.099102	192.168.10.98	192.168.10.17	SSL	590	Continuation Data

Obr. 1.10 Výpis ze souboru uzel-0-0-.pcap

V konzole se objeví následující výpis, který informuje o počtu odeslaných a doručených paketů, o zpoždění (delay), jitter a datovém toku. Obr. 1.11.

```
Flow ID: 1
Tx Bytes: 619380
Rx Bytes: 619380
Tx Packets: 1100
Rx Packets: 1100
Mean Delay: 493.057 ms
Mean Jitter: 7.8951 ms
Throughput: 490.865 Kbps
```

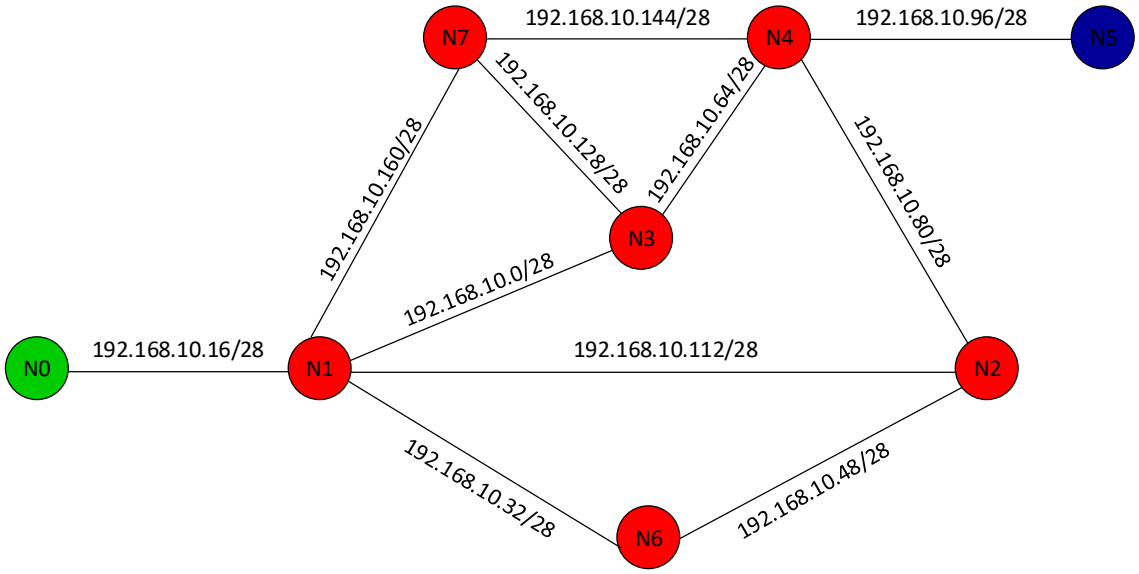
```
Flow ID: 2
Tx Bytes: 28656
Rx Bytes: 28656
Tx Packets: 551
Rx Packets: 551
Mean Delay: 3.37545 ms
Mean Jitter: 0.000454553 ms
Throughput: 22.7127 Kbps
```

Obr. 1.11 Prenos pomocou TCP protokolu

4. Prozkoumejte zprávy na rozhraní 0 uzlu 0 (`uzol-0-0.pcap`). Všimněte si rozdíl mezi UDP a TCP protokolem. Při TCP protokolu najděte three-way handshake a four-way handshake. Nachází se záložce `Statistics`,

kde vybereme položku `Flow Graph`. Otevře se nové okno, ve kterém můžeme vidět, jak probíhala komunikace mezi uzlem N5 a N0.

5. Do topologie přidejte uzel N7. Uzel N7 bude spojen s uzly N1, N3 a N4 linkou point-to-point. IP adresy rozhraním nastavte podle obrázku Obr. 1.12, metriky všem rozhraním obnovte na hodnotu 20. Nezapomeňte nastavit vhodné souřadnice uzlu N7 pro animaci podle zobrazený topologie.

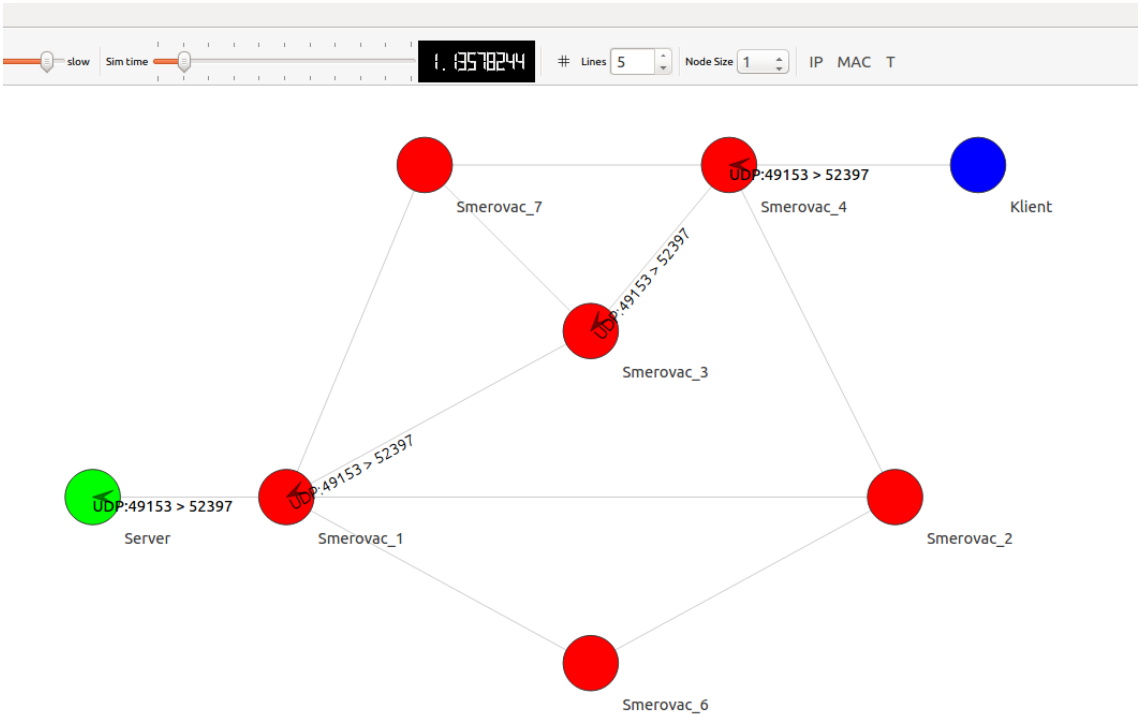


Obr. 1.12 Výslední topologie laboratorní úlohy

6. Nastavte výpadek linky mezi uzly N2, N4 v čase 7s (vypněte rozhraní na uzlu N2 směřující k uzlu N4).

7. Vytvořte aplikaci, která vypíše směrovací tabulky pro čas 9s.

8. Zkontrolujte správnost přidaných linek v programu NetAnim. V době 0-5s směrování probíhá přes směrovače N4-> N3-> N1, v čase 5-7s směrování probíhá přes směrovače N4-> N2-> N1 a v čase 7-10s směrování probíhá přes směrovače N4-> N7-> N1. Výslední topologie z programu NetAnim v čase 8s na obrázku Obr. 1.13.



Obr. 1.13 Výslední topologie z programu NetAnim

1.7 Kontrolní otázky

1. Jaký je rozdíl mezi link-state a distance-vector protokoly?
2. Podle čeho probíhá výběr nejlepší cesty v protokolu OSPF?
3. Jaké čísla portů mají služby TFTP a HTTPS. Jak rozdělujeme porty?
4. Jaké zprávy se vyměňují při sestavování a ukončování spojení při protokolu TCP?
5. Proč je velikost toku přijatých a odeslaných dat rozdílná u protokolu TCP?
6. Jakou velikost mají posílány pakety při protokolu UDP a jakou při protokolu TCP?
7. Jaký je nástupce protokolu SSL používaný při protokolu HTTPS?